UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

TRAINING CONVOLUTIONAL NEURAL NETWORKS FOR IMAGE CLASSIFICATION WITH QUALITY-REDUCED EXAMPLES

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

PABLO FELIPE TORRES GUTIÉRREZ

PROFESOR GUÍA:
AIDAN HOGAN

MIEMBROS DE LA COMISIÓN:
JUAN MANUEL BARRIOS NÚÑEZ
FELIPE BRAVO MÁRQUEZ
PABLO ROMÁN ASENJO

Este trabajo ha sido parcialmente financiado por:
IMFD

SANTIAGO DE CHILE
2022

ENTRENAMIENTO DE REDES NEURONALES CONVOLUCIONALES PARA CLASIFICACIÓN DE IMÁGENES CON EJEMPLOS CON CALIDAD REDUCIDA

En tareas de Visión por Computadora, las Redes Neuronales Profundas pueden superar a los clasificadores humanos en la tarea de Clasificación de Imágenes. Por lo general, se entrenan los clasificadores con imágenes de entrenamiento y conjuntos de validación sin perturbaciones en su calidad. En este contexto, queremos estudiar el comportamiento de las Redes Neuronales al entrenarlas con imágenes con calidad reducida.

Estudios previos han reducido la información necesaria de una imagen para obtener una clasificación correcta utilizando Redes Neuronales, minimizando la entropía o peso de una imagen mediante reducciones de su calidad. La entropía es aproximada con el peso de la imagen en disco, la que depende del tipo de compresión utilizada para la imagen. Esta compresión, si es sin pérdida, tiene como valor mínimo posible la entropía de Shannon.

Las reducciones de calidad antes mencionadas son la aplicación de una transformación a una imagen que disminuye su calidad, por ejemplo recortar la imagen, disminuir su resolución o su cantidad de colores. Este tipo de imágenes plantea un desafío a las Redes Neuronales, pues al tener menor información, puede llevar a una clasificación incorrecta. Al ser entrenada una red neuronal con este tipo de reducciones, podría mejorar su rendimiento sobre imágenes con reducciones de calidad, mejorando así su rendimiento sobre imágenes con distorsiones. En este trabajo proponemos metodologías de entrenamiento para Redes Neuronales en las que se aplican reducciones de calidad para generar imágenes con las que será entrenada la red neuronal.

Evaluamos nuestra propuesta de metodología de entrenamiento sobre el conjunto de imágenes ImageNet y HumaNet (subconjunto que posee imágenes con reducciones de calidad). Los experimentos evalúan el rendimiento y su progresión. Los resultados obtenidos muestran que nuestra metodología puede, en algunos casos, mejorar el rendimiento de las Redes Neuronales en conjuntos de imágenes reducidas y mantener un rendimiento similar al baseline en conjuntos de imágenes sin reducción. Además, las Redes Neuronales entrenadas con nuestra metodología logran clasificar correctamente imágenes con menos calidad.

La principal conclusión es que al entrenar las Redes Neuronales con imágenes con reducciones de calidad, el rendimiento sobre conjuntos con calidad reducida muestra una mejora en algunas arquitecturas de clasificación, en particular el aumento de 71.6% a 72.2% en accuracy de EfficientNet, con respecto a nuestro baseline en el conjunto ImageNet. El trabajo futuro considera la construcción de conjuntos de imágenes con reducciones de calidad, el estudio de este método como forma de compresión y la realización de experimentos con otras heurísticas.

TRAINING CONVOLUTIONAL NEURAL NETWORKS FOR IMAGE CLASSIFICATION WITH QUALITY-REDUCED EXAMPLES

In Computer Vision tasks, Deep Neural Networks may surpass humans classifiers in the Image Classification task. These classifiers based on Deep Neural Networks are typically trained assuming ideal conditions for training images and validation sets, namely, images without loss of quality. In this context we want to study the behavior of Neural Networks when trained over reduced-quality images.

Previous works have reduced the necessary information of an image for obtaining a correct classification using Neural Networks by applying quality reductions. The information is approximated as the size in disk of the image, which depends on how the image is compressed. This compression, if it is a lossless compression, has the Shannon entropy as its minimum possible value.

The aforementioned quality reductions are the application of a transformation on an image which decreases its quality, for example, to slice an image, to decrease its resolution or its amount of colors. These reduced types of images present a challenge to Neural Networks, where less information can result in an incorrect classification. On the other hand, these types of reductions, applied during training of a Neural Network, may improve its performance over quality-reduced images, thus improving its performance for distorted images. In this work we propose training methodologies for Neural Networks in which quality reductions are applied to generate reduced images that will be used to train the classifier.

We evaluate our proposed training methodologies over datasets of images: ImageNet and HumaNet (which is a subset of ImageNet that also contains quality-reduced images). Experiments evaluate the performance of the classifiers as they are trained. The obtained results show that our methodology, in some cases, improves the performance of Neural Networks on images with quality reductions while keeping a similar performance with respect to the baseline in images without quality reductions.

Our principal conclusion is that training Neural Networks with quality-reduced images shows an improvement over both quality-reduced and full-quality datasets in some classification architectures, in particular, the accuracy improves from 71.6% to 72.2% for EfficientNet with respect to our baseline (training only with full quality images) considering the ImageNet dataset. Future work includes making novel datasets with quality-reduced images, the study of this method as a form of compression and experiments with other heuristics.

A la voluntad de crear,
soñar,
avanzar,
y ser feliz

Agradecimientos

Un sueño que se convirtió en motivación, un camino y también, un desafío. Quiero agradecer a mi Papá Patricio, a mi Mamá Catherine y a mi Hermano Cristóbal por sus distintas formas de apoyarme en este sueño.

Agradezco a mi Tío Luis, Tía Pilar, Tía Myriam, Primos Nacho y Javi, por haberme acogido como si fuese uno más de la Familia, al haber llegado a la capital. Agradezco a mi Profesora de Inglés Miss Carolina Guajardo y Profesor de Física Sergio Neira por su motivación, enseñanzas y consejos.

Agradezco al Profesor Sergio Ochoa por el aprendizaje durante el tiempo que fue cliente de mi equipo de Ingeniería de Software II y por sus consejos con respecto al mundo académico. Agradezco a la Profesora Andrea Rodríguez Silva por haber confiado en mí para ser parte del equipo docente en el curso de Herramientas para el Trabajo en Equipo. Ambos cursos han sido muy útiles para mi formación y me alegra haber podido compartir con ustedes.

Agradezco a mi Maestro y Profesor Guía: Aidan Hogan, su apoyo durante este desafío ha sido constante y en cada reunión he logrado aprender sobre la metodología y técnicas para realizar investigación en Ciencias de la Computación. También le agradezco por los cursos electivos que dictó pues aprendí herramientas que permiten hacer realidad la tecnología del futuro.

Agradezco a mi Hermano, mi Padawan: Cristóbal Torres Gutiérrez. Hermano gracias por el apoyo durante el desarrollo de la Tesis; me alegra haber compartido y discutido formas de llevar a cabo los problemas; la ayuda para comprender algoritmos y metodologías. De verdad sentí como si fuesemos un gran equipo y me alegra mucho que hayas escogido también este apasionante mundo de la tecnología como profesión.

Agradezco a los Profesores con quienes discutí durante el desarrollo de la Tesis: Jorge Pérez y José Manuel Saavedra, gracias por su aporte. Agradezco a Pablo Pizarro por su template de Latex y su ayuda para comprender sus funcionalidades. Agradezco al Profesor Adolfo Carrasco por sus consejos y técnicas de presentación. Agradezco a los integrantes de mi comisión: Juan Manuel Barrios, Felipe Bravo y Pablo Román por sus comentarios y aportes. Agradezco a quienes conocí durante mis estudios, a las nuevas amistades, sus historias y su apoyo.

Durante la carrera he aprendido mucho más de lo que creía inicialmente, nuevas áreas como el Desarrollo de Software, la Inteligencia Computacional, la Seguridad Computacional, la Investigación en Ciencias de la Computación y también el Desarrollo de Videojuegos. Estoy agradecido de haber tomado el desafío de estudiar en Beauchef.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1. Motivation

Deep Neural Networks are powerful Machine Learning classifiers that achieve excellent results in many research areas, such as visual oriented tasks (Computer Vision) and text-oriented tasks (Natural Language Processing). This learning power has been improved by large amounts of data (e.g. pictures, videos, audio, text).

In the Computer Vision field a task of interest is Image Classification whose goal is to assign classes to unseen images based on what they depict, being an important component for applications such as medical imaging, self-driving cars, biometric recognition (e.g face, fingerprints) and many others. Deep Neural Networks, in particular Convolutional Neural Networks, have led to major advances in the area. In 2012 Krizhevsky et al. [1] popularized the use of convolutions and the power of GPUs to design and train Deep Neural Networks, proposing AlexNet. Architectures such as VGG [2], InceptionNet [3] and ResNet [4] increased the number of layers of the Neural Networks, and succeeded in outperforming AlexNet [1]. SqueezeNet [5] focused on reducing the number of parameters, but preserving the performance of AlexNet. EfficientNet [6] improved performance and also reduced the number of parameters with respect to state-of-the-art Convolutional Neural Networks through balancing depth, width and resolution of the model.

Works in this field have compared the robustness of these algorithms with human vision in the classification of distorted images. Gerihos et al. [7], Dodge and Karam [8] and Carrasco et al. [9] show that humans outperform studied Deep Neural Networks. Carrasco et al. [9] introduces the concept of quality reductions, which consist of transformations applied to an image or matrix to reduce its quality, such as downsampling, quantization and slice. The extent of a reduction can be measured by an appropriate entropy measure, for example, the size of the reduced image on disk. This work also introduces the definition of Minimal-Entropy Positive Image (MEPI), where for a given image, the entropy is minimized subject to a correct classification by the Deep Neural Network.

In order to further increase the diversity and volume of labeled examples available for learning, as well as the robustness of Deep Neural Networks, Data Augmentation techniques can also be applied. These techniques have been used to prevent overfitting in Krizhevsky et al.'s architecture [1] (AlexNet), mitigate class imbalance problems in the work of Chawla et al. [10] and also to improve the robustness of the network using adversarial perturbations in the work by Moosavi-Dezfooli et al. [11]. In particular, robustness against distortions has been achieved by Gerihos et al. [7] showing that the performance of Deep Neural Networks can improve, but with a tradeoff in terms of the generalization of the model when adding new distortions to the images.

However, to date, the literature has tended to focus on using a particular change in the data in order to augment the training set independently of the classifier, rather than studying the effect of different changes for a given classifier. This moves us to a key question: can we improve the performance of a given classifier further by reducing the information (or quality) present in the training examples?

This work will focus on the analysis of the performance of a state of the art Deep Neural Network. The training will consist of a full size image which will be reduced during training in order to obtain a minimal reduced image such that the classifier yields a correct answer, and then use it to train the Neural Network, applying this process recursively to improve the performance of the Neural Network.

1.2. Hypothesis

Our hypothesis is that the classification performance of images (from the ImageNet dataset) of a Convolutional Neural Network trained on a reduced entropy dataset is better, in terms of accuracy, than when it is trained on a dataset without reductions, in two cases:

- With respect to a test set of complete images (without reductions).
- With respect to a test set of images with reductions.

1.3. Objectives

1.3.1. General Objectives

Design a training process that reduces the quality of input labeled examples in order to improve the performance of state-of-the-art Deep Neural Networks for Image Classification on both full quality images and reduced-quality images (or ascertain why this approach does not work).

1.3.2. Specific Objectives

- Specific Objective 1: Establish a baseline considering an image dataset, Convolutional Neural Networks and accuracy as an evaluation metric.
- Specific Objective 2: Design and implement a training methodology for a quality-reduced input.
- Specific Objective 3: Compare the results of the baseline of state-of-the-art Neural Networks (trained on full-quality images) with respect to performance trained on reduced-quality images using the aforementioned methodology. The comparison should include both full-quality and reduced-quality test sets of images.

1.4. Problem

The problem we want to address is the robustness of the performance of Convolutional Neural Networks for Image Classification under image distortions, which may be due to poor weather conditions, poor lighting conditions, poor focus, partially obscured objects (e.g., people wearing masks), etc. Distorted images used as input to systems that implement Computer Vision algorithms may yield an incorrect answer. We also believe that the methodology proposed here can improve Image Classification for full-quality images, leading to more accurate results.

1.5. Contribution

Previous works [7–9] comparing the classification performance of humans and Computer Vision algorithms on distorted images observe that humans outperform such algorithms. To improve the robustness of these algorithms we will train Convolutional Neural Networks with quality-reduced images using specialized training methodologies that consider the feedback from the model for improving its robustness and performance. While other works have considered specific Training Methodologies [1, 7], or Data Augmentation [10–13] techniques for training Neural Networks, to the best of our knowledge, these works have primarily focused on adding specific, independent types of noise, distortion or quality-reductions to images. Such works typically do not use feedback from the classifier during training in order to select the level of distortion, reduction, etc. In our framework, following the work of Carrasco et al. [9], we design a training methodology that unifies different types of quality reductions under the common framework of entropy (estimated as compressed file-size). Carrasco et al. [9] used these reductions to compare human and algorithmic performance, but used Convolutional Neural Networks trained on full-quality images only. Basing our framework on entropy allows us to combine, for example, crop, resolution reduction, color reduction, etc., where the level of reduction is guided by the resulting file-size of the image independently of the type (or combination) of reduction applied. The reduction of training images is further guided by the current performance of the classifier. Thus, for example, our methodology can produce an image for training that combines partial crop, partial resolution reduction and partial color reduction, such that the image is at the limit of what the classifier can classify correctly (in terms of quality/entropy). We hypothesize that this will lead to improved performance for Image Classification using Convolutional Neural Networks on both full-quality and reduced-quality images.

1.6. Methodology

1.6.1. Related Work Survey

In the first phase of this project, we will study state-of-the-art Convolutional Neural Network architectures, Data Augmentation techniques, entropy measures, and training methodologies for the Image Classification task.

1.6.2. Design and Implementation of Algorithms

The next step will be to design an algorithm for training a Deep Neural Network over a large dataset of images, while applying quality reductions on the input, which will be used for training.

1.6.3. Experimentation

The Neural Network, considering the quality reductions on the input images and their reduction parameters, will be compared with a baseline's (the same Neural Network trained with images without quality reductions) performance metrics.

1.6.4. Technologies

The algorithms will be implemented in Python, using Google's Colab service and, for massive data training, an external cluster will be used. The Neural Networks will be implemented in Pytorch.

1.7. Summary of Results

In this work we studied the effect of training Convolutional Deep Neural Networks with quality-reduced images on the performance of Image Classification. Performance was measured in accuracy and entropy-ratio (as Carrasco et al. [9] defined) compared with our baselines. We obtained an improvement from 71.6% to 72.2% in accuracy with respect to the baseline with the EfficientNet architecture on the ImageNet dataset. Other architectures did not show improvement. In the subset of ImageNet (HumaNet) we performed experiments of our Training Methodologies. We saw an improvement of the performance in the quality-reduced datasets depending on the methodology and quality-reduced images used as input. For example, considering EfficientNet, we obtained an improvement in accuracy from 31.0% to 47.7% over images reduced by crop when training likewise over images reduced by crop (and not only full-quality images).

1.8. Work Structure

The present work has the following chapters:

- In Chapter 2 we describe the key theoretical concepts relevant to this work. This chapter first provides an overview of Digital Image Processing, specifically how images are interpreted by the computer and the application of filters over them. We also discuss Probability, Statistics, Information Theory and Optimization Methods for training Neural Networks. Then, we provide an overview of Machine Learning and we describe the training process of Deep Neural Networks as a subset of current Machine Learning algorithms.

- In Chapter 3, we discuss works related to Data Augmentation, the work of Carrasco et al [9] that forms the basis of the theoretical framework of this work, an overview of the Image Classification task, and the performance achieved for this task by Deep Neural Networks.

- In Chapter 4, we give an overview of the proposed training methodology of Neural Networks with quality-reduced examples. This chapter includes the justification of the method, an explanation of the training pipeline, the implemented quality reductions, and the description of each module involved in the training.

- In Chapter 5, we explain the experiments run in this work. We present the research questions, the description and justification of the Neural Networks architectures and datasets used, the baseline we compare our results with, and the metrics used to quantify the performance of each methodology.
- In Chapter 6, we present the obtained results of the experiments run. We also provide a discussion and analysis of the results.
- In Chapter 7 we conclude our work, discussing the limitations and future work for this proposed methodology.

Chapter 2

Background

In this chapter, we introduce some concepts that are foundation to our work. In Section 2.1 we introduce key concepts to understand how images are interpreted by computers and the transformations applied to them (to later understand the application of quality reductions). In Section 2.2 and Section 2.3 we offer a refresher on Probability and Statistics and Information Theory, introducing two key concepts for choosing the loss function used in the training process of Deep Neural Networks: Maximum Likelihood Estimation method and Kullback-Leibler Divergence. In Section 2.4 we discuss the optimization methods used in the training process of Deep Neural Networks and Powell's Method used in the work of Carrasco et al. [9]. In Section 2.5 we present an introduction to Machine Learning. We then explain Neural Networks and their training process in Section 2.6. Finally in Section 2.7 we introduce the Image Classification Task, Training Methodologies and the Convolutional Neural Network Architecture.

2.1. Digital Image Processing

In this subsection we present how digital images can be interpreted, manipulated and represented by computers; their unit expression known as a pixel; the color representation in the convention model known as RGB; the concept of image compression; the image format provided by the ImageNet and HumaNet datasets; and finally, computational applications over images.

2.1.1. RGB Model

The RGB model is an additive color model in which each letter represents Red, Green and Blue, respectively, as the primary colors of light. These colors, added together, can reproduce a broad composition of colors. For example the combination of all three colors in equal intensity makes the color white.

2.1.2. Pixel

In digital imaging the pixel is the smallest addressable element in a raster image; this implies that it is the smallest controllable element of a picture represented on a display device. The intensity of each pixel is variable (for example, by regulating the intensity of a red light represented by a pixel).

In terms of the RGB color model, each pixel has a discrete finite value representing its intensity, which can take values between 0 and 255 (minimum and maximum intensity, respectively). In a three-channel model, where each channel represents one color of the RGB color model, the combination of each one at maximum intensity (represented as (255, 255, 255)) makes the color white.

2.1.3. Raster graphics

Raster graphics is a mechanism that represents a two-dimensional image as a rectangular matrix or grid of square pixels viewable via a computer display. It is characterized by the width and height of the image in pixels and by the number of bits per pixel.

2.1.4. Resolution

In digital imaging, resolution is equivalent to the total count of pixels. The convention is to describe the pixel resolution with two positive integer numbers: the first is the number of columns and the second the number of rows if we represent the image as a matrix. These numbers define the dimension or size of the image.

2.1.5. Compression

Image compression is a type of data compression which is applied to digital images; it reduces the cost of storage or transmission. There are two types of image compression: lossy and lossless.

2.1.5.1. Lossy Compression

Lossy compression uses inexact approximations to represent the compressed content that may discard some data. The JPEG image format is an example of this type of compression based on the discrete cosine transform method [14]. Well-designed lossy compression methods can reduce file sizes significantly before degradation is noticeable by the end-user.

2.1.5.2. Lossless Compression

Lossless compression allows the original image to be completely reconstructed from the compressed data. This type of compression is used in cases where it is important that the original and the decompressed image be identical. An example of image extension that is compressed using this method is PNG.

2.1.6. Digital Image

A digital image is an image composed of pixels, each with finite, discrete quantities of numeric representation for its intensity or gray level, which maps to a plane with an x-axis and a y-axis. When the RGB color system is applied the image is represented by a three dimensional matrix in which the first and second dimension represents a pixel and the third dimension the channels that capture each color of the RGB color system (Red, Green and Blue).

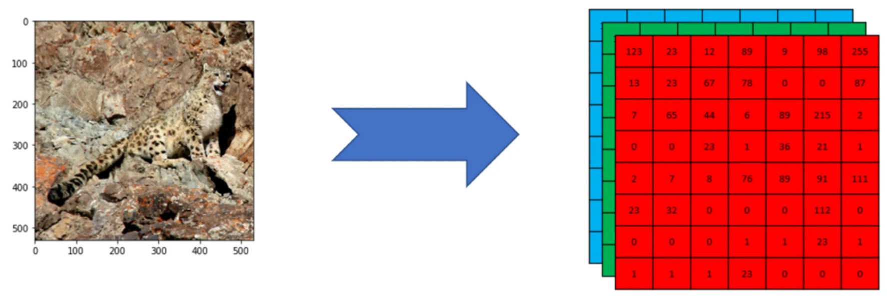


Figure 2.1: Matrix representation of a digital image. Source 'Educative.io'.

2.1.7. Spatial Filtering

Spatial filtering is a technique used to modify or improve an image. For example it can emphasize certain regions of an image (e.g. edges) or erase other entities (e.g. certain colors). Its application requires an image, a region of an image $I(x, y)$ and an operation made in that region. The operation is performed by a mask known as a kernel.

2.1.7.1. Correlation Filtering

We define the linear filtering for a given image $I_{M\times N}$ and a kernel $W_{K_y\times K_x}$ by the following mathematical expression:

$$G(y,x) = \sum_{v=-r_y}^{r_y} \sum_{v=-r_x}^{r_x} I(y+v, x+u)W(v,u) \tag{2.1}$$

As an example we consider the following image I with the kernel W (we make the assumption that outer bounds operations are completed by zeroes), where we compute the resulting image $G = I \bigotimes W$:

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, W = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, G = I \bigotimes W = \begin{bmatrix} 3 & 5 & 3 \\ 9 & 11 & 6 \\ 15 & 17 & 9 \end{bmatrix} \tag{2.2}$$

2.1.7.2. Convolution Filtering

We define the convolution filtering for a given image $I_{M\times N}$ and a kernel $W_{K_y\times K_x}$ by the following mathematical expression:

$$G(y,x) = \sum_{v=-r_y}^{r_y} \sum_{v=-r_x}^{r_x} I(y-v, x-u)W(v,u) \tag{2.3}$$

It is similar to linear filtering, but the kernel is reflected with respect to the center as observed by the sign of v and u.

2.2. Probability and Statistics

Probability theory represents uncertain statements and quantifies this uncertainty. In Machine Learning probability is applied to understand how algorithms should make decisions and to analyze their behavior theoretically. In this subsection we present key concepts relating to probability and statistics that are applied in the learning process of Neural Networks.

2.2.1. Probability Axioms

A probability $\mathbb{P}$ is a function of real values defined over Ω that satisfies the following axioms:

- Axiom 1: For any event $E \subseteq \Omega$, $0 \leq \mathbb{P}(E) \leq 1$.
- Axiom 2: The probability of the sample space Ω is 1.
- Axiom 3: Let $E_1, ..., E_k \in \Omega$ be disjoint sets: $\mathbb{P}(\bigcup_{i=1}^{k} E_i) = \sum_i^k \mathbb{P}(E_i)$ holds.

2.2.2. Random Variable

A random variable is a function $X : \Omega \rightarrow \mathbb{R}$ which maps any event of $e \in \Omega$ to a real value $X(e)$. For example we can model the toss of a coin with outcomes tail (T) and heads (H) and define $X(e)$ as the number of heads in a sequence of tosses; for example, if $e = HHHTHTTH$, then $X(e) = 5$, because in the sequence we have five values of H.

2.2.3. Probability Distribution

A probability distribution is a function that gives the probability of occurrence of different possible outcomes for an event. Its description depends on whether the described variable is discrete or continuous.

2.2.3.1. Normal Distribution

The normal distribution is a continuous distribution and its mathematical expression represents μ as the mean or expectation of the distribution and σ is its standard deviation; it has the following expression:

$$f_x(\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}} \tag{2.4}$$

2.2.4. Statistical Inference

Statistical inference is the process of drawing conclusions from samples of a population. Its goal is to infer the distribution that generates the observed data. The models that assume a distribution can be described by a set of parameters $\theta = (\theta_1, ..., \theta_k)$ and are called parametric models. There are two approaches for making inferences: frequentist and Bayesian inference. For example in the case of a set of coin tosses the probability of success p is a parameter that can be inferred by a $\hat{p}$ estimated parameter.

2.2.5. Maximum Likelihood Estimation

Maximum Likelihood Estimation is a method which estimates the parameters of an assumed probability distribution, given some observed data. This estimation is achieved by maximizing a likelihood function so that, under the assumed probability distribution, the observed data is most probable.

In the following equation $\mathcal{L}_n$ is the likelihood equation in which f is a known probability distribution, X_i are independent and identically distributed random variables, and its probability density function is $f(x;\theta)$. This function is maximized to get the parameters that best fit the data.

$$\arg\max_{\theta} \mathcal{L}_n(\theta) = \arg\max_{\theta} \prod_{i=1}^{n} f(X_i;\theta) \tag{2.5}$$

One property of this estimator is consistency. This means that if the data were generated by $f(x;\theta)$ and with a large number of observations n, it is possible to find the value θ_0 with arbitrary precision. Mathematically it means that the estimator converges in probability to the true value of the distribution $\hat{\theta} \to \theta_0$.

2.2.6. Loss Function

A loss function or cost function is a mathematical function that maps values of one or more variables to a real number that represents the cost of a certain event. In statistics a loss function is typically used for parameter estimation, and the event in question is some function of the difference between estimated (or predicted) values and true values for an instance of data. An optimization problem seeks to minimize a loss function.

2.3. Information Theory

Information Theory is the study of quantification, storage and communication of digital information which, in the presence of uncertainty, allows us to quantify the amount of uncertainty in a distribution of probability.

2.3.1. Entropy

Considering a group of symbols and the discrete density function of each symbol, with a probability p_i for an event i the Shannon entropy H, in units of bits (if we consider a base two for the communication compression), has the following mathematical expression:

$$H = -\sum_{i} p_i \times \log_2(p_i) \tag{2.6}$$

Considering a discrete random variable X, this expression indicates the amount of necessary bits to represent X when only its distribution is known.

2.3.2. Kullback–Leibler Divergence

Defined by Kullback and Leibler [15], the divergence of information, also known as relative entropy, measures the extent to which two probability distributions Q and P, defined on the same probability space $\mathcal{X}$, are different, taking P as the reference probability distribution. It is interpreted as the excess of information content (a basic quantity derived from the probability of a particular event occurring from a random variable) from using Q as the model when the actual distribution is P. This divergence has the following mathematical expression:

$$D_{kl}(P,Q) = \sum_{x\in\mathcal{X}} P(x)\log\left(\frac{1}{Q(x)}\right) - \sum_{x\in\mathcal{X}} P(x)\log\left(\frac{1}{P(x)}\right) \tag{2.7}$$

2.4. Optimization Methods

Optimization methods are applied to get the best possible solution to a given mathematical problem, which may be the maximum or minimal solutions for the parameters in study. In Deep Learning, optimization methods are crucial for finding the best parameters of the Neural Network. Such methods were also used by Carrasco et al. [9] for finding the parameters that minimize the entropy of a given image while yielding a correct classification, i.e., for finding the *Minimal Entropy Positive Image.*

2.4.1. Stochastic Gradient Descent

Stochastic Gradient Descent is a method for optimizing an objective function through iterations of an approximation of the gradient. This method reduces the high computational complexity of high-dimensional optimization problems in order to achieve faster iterations with the trade-off of a lower convergence rate.

The parameters or weights of the model are updated for a given function $Q(w)$ in which each observation i is associated with its summand function $Q_i(w)$ computing the gradient $\nabla Q_i(w)$ of each one and multiplying with a factor η known, in the Machine Learning terminology, as *learning rate.* The iterations made are represented by the following expression:

$$w_{i+1} := w_i - \eta\nabla Q(w_i) \tag{2.8}$$

This method also has improvements and extensions for making the learning rate adaptive, for addressing high divergence (if the learning rate has a high value) and for addressing low convergence (if the learning rate has a low value). Some examples of these extensions are Momentum, RMSProp [16] (Root Mean Square Propagation) and ADAM [17] (Adaptive Moment Estimation).

2.4.2. Powell's Method

Powell's method [18] is an algorithm for finding a local minimum of a function without the condition of the function being differentiable, because it does not compute derivatives during the search. It proceeds as follows:

Let us consider a starting point $x_0 \in \mathbb{R}^n$ and a list of N search vectors $S := \{s_1, ..., s_N\}$. The algorithm proceeds to minimize the objective function by a *bi-directional* search along each search vector $s_i \in S$. Then, the *bi-directional* search produces the following vectors $S' := \left\{x_0, x_0 + \alpha_1 s_1, ..., x_0 + \sum_{i=1}^{N} \alpha_i s_i\right\}$, where every α_i is obtained by the *bi-directional* search along s_i. Then, the new position obtained is defined as $x_1 := x_0 + s'_i = x_0 + \sum_{i=1}^{N} \alpha_i s_i$ with $s'_i \in S'$, and s'_i added to the list of search vectors S. Next, the search vector that contributed the most for the direction of s'_i is erased from that list, and its position of the list is defined as $i_d = \arg\max_i^N |\alpha_i||s_i|$, thus the updated search vector set is: $S := \{s_1, ..., s_{i_d-1}, \cancel{s_{i_d}}, s_{i_d+1}, ..., s_N, s'_i\}$. The algorithm continues until the maximum number of iterations is reached.

2.5. Machine Learning

Machine Learning proposes a set of computer algorithms, which improve automatically through experience or training. In the Image Classification task, state of the art models are Deep Neural Networks, which have been more popular in recent years due to advances in GPU hardware technology and the wider availability of data. We now describe these concepts in more detail, as relevant for this thesis.

2.5.1. Supervised Learning

Supervised Learning is the task of learning a function that maps an input to an output value or vector. The function is inferred from labeled training data and then evaluated in test data. It is called supervised because the process of learning or inferring the mapping function is actively supervised by the algorithm as it receives feedback of the results that the current mapping outputs.

2.5.2. Performance Metrics

A performance metric is a value that qualifies or judges the performance of a Machine Learning model. A simple metric to use is classification accuracy, which is defined as the number of correct predictions divided by the total number of predictions.

2.6. Neural Networks

A Neural Network is a Machine Learning model inspired by biological neural networks composed of neurons that communicate via specialized connections called synapses between neuron dendrites. A simple artificial Neural Network model receives an input in its input layer in the form of a vector; then it processes the data and the results obtained by the hidden layers, operations are committed in the output layer. Every layer has a minimal unit, called a neuron, which simulates a biological one.

Each neuron receives a signal from the previous layer that is weighted by a weight value which represents the intensity of the connection between neurons. Then the activation of the neuron depends on whether or not the sum of the signal received is superior to a threshold called bias. Nevertheless, the model cannot solve complex problems this way, because it is equivalent to a linear model, so a non-linear activation function is typically added to allow non-linear classifications. In Figure 2.2 we show a representation of a Neural Network with two hidden layers.

Due to better hardware (GPUs) and the birth of other Neural Network architectures, nowadays the number of hidden layers have increased and this increase led to Deep Neural Networks.

In this subsection we describe the process of training a Deep Neural Network and the regularization methods applied in this process.

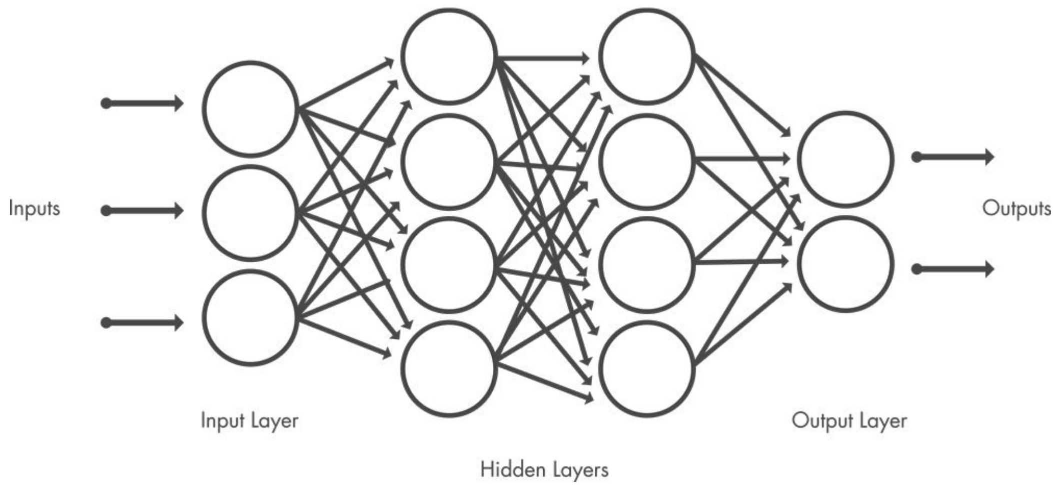


Figure 2.2: Neural Network with two hidden layers. Source 'Wikimedia Commons'.

2.6.1. Neural Network Training

The phases of Neural Network Training can be divided in two: the forward and the backward process. In the former, the Neural Network, given an input, provides results of a certain classification and the latter corrects and adjusts the network parameters according to the loss function and the optimization method for the training process. These processes will be discussed in the next subsection with its components.

2.6.1.1. Forward

In this step the input vector is processed by the network with its internal operations (in the hidden layers) until it reaches the output layer. Then the classification result is the argument of the maximum value of the output vector (i.e., its index in the output vector).

For training, the softmax function is applied to the output vector to associate the vector to a probability distribution of the results (this probability notion allows us to compute a maximum likelihood estimation of the data given the parameters). This value is then passed to the loss function which compares the real result with the predicted result in the loss function to then, in the backward process, update the weights of the Neural Network.

2.6.1.1.1. Activation Functions

An activation function is a mathematical function that defines the output of a node of the Neural Network that will be passed to the next layer. For the Neural Network to represent a wide variety of functions it is necessary to use a non-linear activation function. If this property suffices the Neural Network can become a universal function approximator.

Some examples of these activation functions are RELU (Rectified Linear Unit), Sigmoid and ELU (Exponential Linear Unit).

2.6.1.2. Softmax

Softmax is a mathematical function which takes as input a vector $v \in \mathbb{R}^K$ and then it normalizes it into a probability distribution of K values in the interval $(0, 1)$ proportional to the exponential of the vector values; this function is interpreted as a "smooth argmax". Each output value is obtained by the following expression:

$$\frac{e^{v_i}}{\sum_{j=1}^{K} e^{v_j}} \tag{2.9}$$

2.6.1.3. Backward

In the backward process we compute the derivatives of each weight according to the results obtained during the forward process using the backpropagation algorithm. Then we update each weight of the Neural Network using the optimization function.

2.6.1.3.1. Backpropagation

The backpropagation algorithm computes the gradient of the loss function with respect to each weight of the Neural Network by the chain rule. It uses dynamic programming to pre-calculate the expression of the derivative, which is then computed iteratively from the last layer calculating the gradient of each weight.

2.6.1.3.2. Loss function

The loss function is the function from which the derivatives are computed to update the parameters by minimizing the loss; thus, it defines an objective to evaluate the model. Cross-entropy Loss is often used for classification tasks, defined as it follows:

$$\mathcal{L} := -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(\hat{y_i}) + (1 - y_i) \cdot \log(1 - \hat{y_i}) \tag{2.10}$$

Where N is the number of samples, y_i is the known class label, and $\hat{y_i}$ is the probability of each class label predicted by the model.

Minimizing this loss function is equivalent to maximizing the likelihood of the parameters, so it has the same properties as the maximum likelihood estimator, which was defined in Section 2.2.5. This minimization is also equivalent to minimizing the Kullback-Leibler Divergence for obtaining the parameters for the probability distribution, defined in Section 2.2.5.

2.6.1.3.3. Optimizer

An optimizer is a method used to update the weights of a Neural Network, minimizing the loss function according to the derivatives computed on the backpropagation algorithm step. For a given model and its parameters it updates them according to Section 2.4.1.

2.6.2. Regularization Methods

Regularization methods are techniques for improving the generalization (i.e., the ability of the model to adapt to previously unseen data) of the Neural Network model. Some of the described techniques are compared and discussed by Bengio [19].

2.6.2.1. Weight Decay

Weight decay is a regularization technique that adds a penalty term to the cost function of a Neural Network training; its effect is to shrink the weight's values during the backward process. This helps the Neural Network to prevent overfitting (which is when the model cannot generalize to unseen data).

2.6.2.2. Early Stopping

Early stopping is a regularization technique used to avoid overfitting during training. This method provides rules to guide how many iterations can be run without performance improvement before the learning algorithm begins to over-fit. The number of epochs defined to wait before stopping the training process early is defined as patience. This value is a hyperparameter and different works use different values of it, for example, Wei et al. [20] use a value of 3.

2.6.2.3. Data Augmentation

Data Augmentation is a regularization technique, which increases the amount of data by adding modifications to existing data (e.g. images or text); thus it creates synthetic data from the original or existing data.

2.6.3. Hardware and Libraries

In this subsection we introduce the hardware that has allowed Deep Neural Networks to achieve results efficiently (in terms of time). We also introduce a library for the Python Programming Language, which allows us to communicate and work with this hardware.

2.6.3.1. GPU

A Graphic Processing Unit is a processing hardware unit, similar to the CPU, but oriented to floating point operations, such as often needed for graphics processing. The main difference between GPU and CPU is the amount of cores a GPU has, where the higher number of cores in a GPU enables high levels of parallelism for computations such as matrix operations. This makes the use of GPU effective for the development of Neural Network models in which matrix operations are made frequently.

Two of the most used Deep Learning libraries, such as TensorFlow and Pytorch, use the GPU's functionalities to improve the performance of Deep Neural Network models. The GPU's API depends on the hardware, and in the case of an Nvidia GPU, it uses the CUDA API to leverage its functionality in the aforementioned libraries.

2.6.3.2. Pytorch

Pytorch is an open source Machine Learning library based on the Torch library available in the Python programming language. Among the advantages of this library is the capacity to process a large volume of data and its ability to use the GPU for tensor computing, which improves speed during the evaluation and training of a model.

2.7. Image Classification

Image Classification has been a key task in Computer Vision, which has been important for learning how human vision works and how to apply it for recognition of images using computer algorithms. In order to standardize the study of this task, the ImageNet Dataset was proposed [21], which has one thousand object classes, including dogs, cats, trucks, ships, etc. This dataset provides a common collection of images to test and compare the different algorithms available for Image Classification.

In the classification algorithms, high-level features are important to classify images, and these features can be extracted manually [22] or automatically by the algorithm. Most modern classification algorithms apply an automatic extraction of features.

With the results of AlexNet in 2012 [1], the Convolutional Neural Network architecture, and different training methodologies over it, leading to continuously improving results on this task in terms of the accuracy metric.

In this subsection we describe the ImageNet dataset and the HumaNet dataset, the Convolutional Neural Network architecture and training methodologies applied for this task.

2.7.1. Dataset

ImageNet is a dataset, which maps objects to different images, organized according to the WordNet (large lexical database of English) hierarchy. ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [21] was an annual challenge (until 2017) that sought to evaluate algorithms for the tasks of image detection and classification. This dataset is still available for measuring and comparing the performance of the developed algorithms on this image task.

HumaNet [9] is a subset of images of ImageNet that serves as a benchmark of the robustness of image classifiers with respect to incomplete information (i.e., with respect to a full quality image).

2.7.2. Training Methodologies

In Image Classification, new Deep Neural Network architectures have been made to improve performance in this task, but also new training methodologies have been developed on these architectures. Here we describe methodologies that have been used in this computer vision task.

2.7.2.1. Generative Adversarial Networks

This methodology trains a Deep Neural Network as a generative model, i.e., a model which generates images that improve during training to the point of being indistinguishable from the real images. This is done by having two models: one which generates images and the second, which discriminates if it is a real or a generated image. These models are called generator and discriminator respectively.

The objective of this training methodology is to improve the generator such that the discriminator does not distinguish between a real and a generated image; for this reason the approach is called adversarial. This architecture can be used to synthesize new data and use it in a data augmentation process [13, 23].

2.7.2.2. Masked Autoencoders

This methodology works by reconstructing masked fragments of an image, so the objective of the model is to learn the fragments that best suit the proposed image with less information. This way the model learns how to yield a correct prediction with less information. This methodology uses the transformer architecture as its backbone (i.e., Neural Network that extracts features).

In Figure 2.3 we show the Masked Autoencoder architecture pre-training pipeline. The input image has a subset of image patches, which are randomly masked out. The encoder receives the subset of visible patches of the image; then the encoded patches and the masked tokens (gray squares) are passed through the decoder reconstructing the input image. For recognition tasks the input is a full-quality image (in this work such images are called uncorrupted images) and the decoder is discarded.

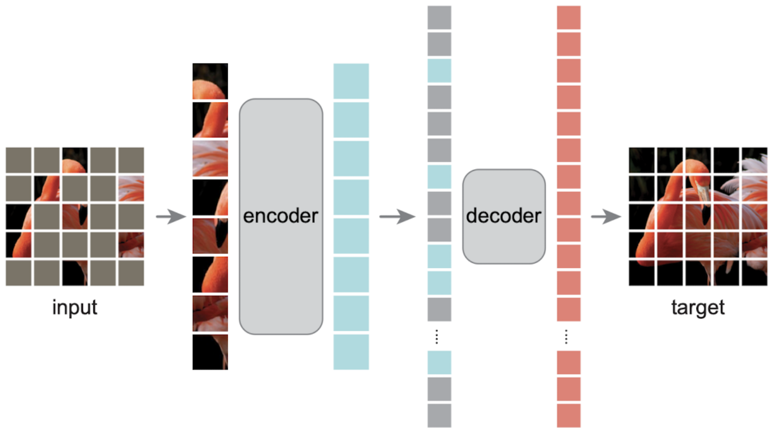


Figure 2.3: Masked Autoencoder pre-training pipeline. Source 'Masked Autoencoders Are Scalable Vision Learners' [24]

2.7.3. Convolutional Neural Network

Convolutional Neural Networks are a Neural Network architecture, which have greatly improved performance on Computer Vision tasks.

The input of the convolutional layer, in the case of the Image Classification task, is a set of the matrix representations of the images stacked in a single tensor called a batch. The dimensions of the input is a tensor of the given dimensions: $B \times C \times H \times W$, which are respectively batch size (amount of images), channels, height and width of each image. Then a convolutional filter is applied to the tensor producing the feature map of the image, whose dimensions depend on the amount of kernels, padding (extra pixels around the boundary of the input image) and stride (number of rows and columns traversed per slide of the kernel).

The following layer is the pooling layer, which reduces the dimension of the data through the combination of the output of neuron clusters in one layer to a single neuron in the next layer. Common pooling operations are: max pooling (takes the maximum value of a region) and average pooling (takes the mean of a region).

Finally the processed data obtained through these operations are connected to a fully connected layer, which takes high level features of the image so it can classify it, learning what feature maps get the best performance for the classification task.

In terms of parameters, given R kernels with dimension $K_y \times K_x$ and an image of C channels, the number of trainable parameters are $C \times K_y \times K_x \times R + R$ in a convolutional layer. A possible interpretation of this process may be that the network learns the best filters to apply with the objective to classify the given images.

In Figure 2.4 we can see the flow of the data from the input image (left) to the output layer (right), passing through convolutional layers, pooling layers and finally a fully connected layer from which the classifier predicts the class of the image.

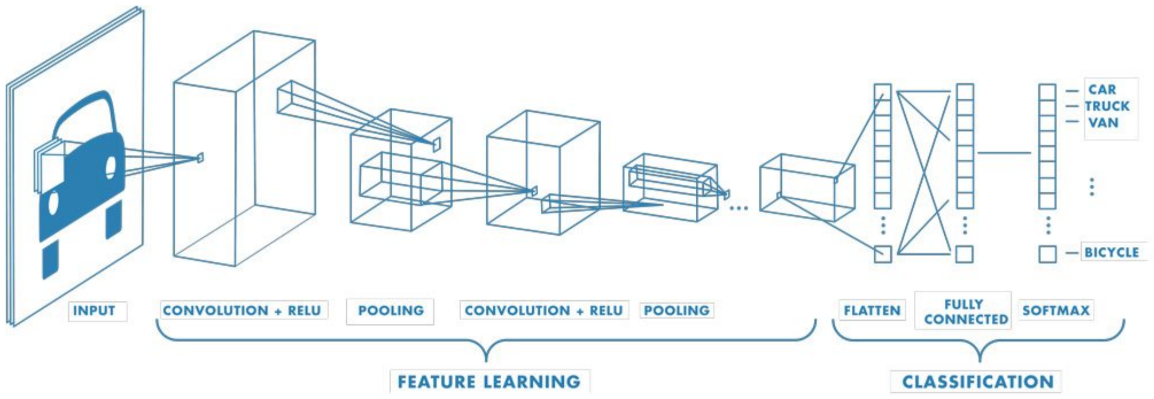


Figure 2.4: Convolutional Layer Source 'Mathworks'.

Chapter 3

Related Work

With an overview of general concepts relevant to this theis, we now discuss in more detail works that are most closely related to this thesis: Data Augmentation techniques and training methodologies for the Image Classification task. The works described in this section form a starting point for the research and development of our work.

3.1. Image Classification

For visual recognition tasks, Convolutional Deep Neural Networks achieve excellent performance. The architectures described here focus on the Image Classification task, which consists of assigning a label to an image or photograph, for example, to classify a handwritten digit.

The work of Krizhevsky et al. [1] describes a methodology for training with a large dataset (such as ImageNet) on a Convolutional Neural Network, considering the effects of Data Augmentation on the performance of the classifier. The proposed architecture (AlexNet) achieved a top-5 error of 15.3% in 2012, popularizing the use of Convolutional architectures for Deep Neural Networks. It also describes services, which can be used for data labeling, such as Amazon Mechanical Turk.

Today, there are Convolutional Deep Neural Networks architectures, which use more layers in their constructions surpassing AlexNet, such as Squeeze-and-Excitation Networks proposed by Hu et al. [25]. This work introduces a building block for Neural Networks that captures channel relationships and dynamically re-calibrates channel-wise feature responses by explicitly modeling inter-dependencies between channels. These blocks can be stacked together to form Neural Network architectures that generalize effectively across different datasets; for example, using these blocks in a Residual Network (ResNet) architecture on the ImageNet dataset achieves a single-crop top-5 validation error of 6.62% (this architecture is called SE-ResNet-50 [25]).

3.2. Data Augmentation Techniques

Shorten and Khoshgoftaar [12] provide a survey that summarizes the techniques for Data Augmentation when training Deep Neural Networks and briefly describe regularization methods on Neural Networks. These techniques are solutions to the problem of overfitting in Deep Learning models due to limited data. Their survey groups the techniques in two classes: Basic Image Manipulation and Deep Learning Approaches. The former considers geometric (e.g. flip, rotate, crop) and photometric (e.g. jittering, random color manipulation, edge enhancement) transformations. The latter considers the use of GANs (Generative Adversarial Networks), Neural Style Transfer, and meta-learning algorithms.

One of the techniques that the survey describes in the second category is the use of adversarial examples that lead state-of-the-art classifiers to give misclassifications where they find the minimum possible noise injection needed to cause such errors (Moosavi-Dezfool et al. [11]). This work is based on an iterative linearization of the classifier to generate minimal perturbations that are sufficient to yield an incorrect classification.

3.3. Robustness of Image Classification

The study of the robustness of Deep Neural Networks in the context of the Image Classification task with respect to the application of distortions on the input image is a key field of study for our work. Hendrycks and Dietterich [26] study the impact of the performance of Deep Neural Networks considering synthetic distortions such as Gaussian noise, zoom blur, rotations, greyscale, pixel noise, etc. Taori et al. [27] study robustness on classifiers considering natural distortions such as lighting conditions, compositions of the scene, the type of the objects, etc.

Comparative works between Deep Neural Networks and humans identifying distorted images have been done to study robustness of these classifiers. Gerihos et al. [7], Dodge and Karam [8] and Carrasco et al. [9] show that humans outperform studied Deep Neural Networks on synthetically distorted images. Gerihos et al. [7] also train the model with images including distortions, showing that the classification of Deep Neural Networks can improve, but with a tradeoff in terms of the generalization of the model when adding new distortions to the images.

3.4. Laconic Image Classification

Laconic Image Classification is a theoretical framework for understanding and comparing classification results based on the principle of computing and analyzing minimal entropy positive inputs [9]: inputs with minimal information with respect to yielding correct classification results. This framework has been used to compare the performance of humans and Deep Learning models in the classification of distorted images, in particular, quality-reductions that yielded less entropy in terms of image size.

A Minimal-Entropy Positive Image (MEPI): is an image $\mathcal{I}'$ that has been reduced, in terms of entropy (with operations called reductions such as crop, downsampling or quantization), from an original image $\mathcal{I}$ (without reductions), such that a given classifier $\mathcal{C}$ (for example a Neural Network) applied to the image $\mathcal{I}'$ gives a correct answer, and for another reduction over $\mathcal{I}'$ the classification yields to incorrect answer.

In other words, for a given machine classifier $\mathcal{C}$, an original image $\mathcal{I}$, a reduction operation $R_i \in \{\text{crop, downsampling, quantization}\}$, a ground truth label for a given image denoted as $\lambda(\mathcal{I})$, reduced images $\mathcal{I}'$ and $\mathcal{I}''$, such that $\mathcal{I} \xrightarrow{R_1} \mathcal{I}_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{n-1}} \mathcal{I}_{n-1} \xrightarrow{R_n} \mathcal{I}_n$, recalling $\mathcal{I}_{n-1}$ as $\mathcal{I}'$ and $\mathcal{I}_n$ as $\mathcal{I}''$, the following condition is satisfied for a MEPI $\mathcal{I}'$: $\mathcal{C}(\mathcal{I}') = \lambda(\mathcal{I})$ (classifier $\mathcal{C}$ evaluated on the image $\mathcal{I}'$ is correct) and $\mathcal{C}(\mathcal{I}'') \neq \lambda(\mathcal{I})$ (it is no longer correct upon further reduction).

For the case of human classifiers, the MEPIs were obtained with a survey made in a web application, in which each participant had to choose between 20 available classes to distinguish a proposed image. The image started from a quality-reduced image. The web application had an option for decreasing the intensity of the reduction, thus improving the quality of the image. When the user felt able to classify the image, they were asked to guess. If they guessed incorrectly, the image was discarded. If they guessed correctly, the image was kept as a MEPI. The difference in obtaining these images between Neural Network models and humans is that in the former the approach was top-down (more quality to less quality) and the latter bottom-up (less quality to more quality).

The final comparison between human and machine classifiers was made obtaining the MEPIs of a subset of ImageNet for each classifier (humans and Neural Network models). Each classifier was evaluated with the MEPIs of the other classifiers obtaining the precision of the classification. These evaluations led to the conclusion that human classifiers were more robust to image distortions compared to the machine models. The obtained MEPIs are available on the internet and the subset was called HumaNet[1].

In this work, we re-use the reductions defined by Carrasco et al. [9]. We made some slight modifications for our Training Methodologies in order to consider quality-reductions as ratios rather than absolute values (with 0 meaning completely reduced and 1 no reduction). In Section 4 we cover the changes made to the quality-reductions of Carrasco et al. [9] and we also show examples for each quality-reduction.

[1] Laconic Image Classification Website: https://aidanhogan.com/laconic

Chapter 4

Proposal

In this chapter the main components of our proposed training methodology are described. First, we provide key definitions for the quality reductions used in this work; afterwards we provide a general overview of the training methodologies proposed.

4.1. Image Quality Reductions

To create labeled examples of images with reduced information for training, quality reductions were implemented using tensor operations provided by Pytorch. These reductions are applied over an image I, in its tensor representation of dimension $C \times H \times W$ (which represent number of channels, height and width respectively) and can be applied to a batch of size B of such images. These image quality reductions are based on Carrasco et al's [9] work using Pytorch tensor operations. We presently provide the mathematical definitions proposed by Carrasco's work, which defines the reduction operations more generally over a matrix $A \in \mathbb{N}^{m \times n}$.

The design of these quality reductions allows us to reduce the size of the images in terms of bytes in a monotonic way, where the entropy of the image is approximated in terms of the byte size of the image after applying lossless compression.

Each reduction requires one or more parameters, which denote the extent of the reduction of the entropy of the image. With more intense quality reduction (as defined here), the image typically requires less size to store in a losslessly-compressed format.

4.1.1. Mathematical Framework

The mathematical framework of the reductions is herein defined over matrices of non-negative integers, but it generalizes to real numbers and other domains. Given an $m \times n$ matrix A, we denote by a_{ij} $(1 \leq i \leq m, 1 \leq j \leq n)$ the element in the i^{th} row and j^{th} column of A. We now define each reduction based on Carrasco et al's [9] quality reductions (we modify some minor details to more easily present our framework later):

- **Quantization**: $A_{\downarrow Q(k)}$ is defined as the nearest value $m \times n$ quantised matrix of A with factor $0 \leq k \leq 1$, such that $a'_{ij} := round(round(ka_{ij})/k)$ for all a'_{ij} in $A_{\downarrow Q(k)}$.
- **Downsampling**: $A_{\downarrow D(s)}$ is defined as the $p \times q$ $(p \leq m, q \leq n)$ downsampled matrix of A with scaling factor s such that $0 < s \leq 1$, $\lfloor sm \rfloor = p$, $\lfloor sn \rfloor = q$, and $p < m$ or $q < n$.
- **Slice**: $A_{\downarrow S(a,b,c,d)}$ is defined as the (contiguous) $p \times q$ submatrix such that $A_{\downarrow S(a,b,c,d)} = (A_{ij})_{\bar{a}m < i \leq m - \bar{b}m; \bar{c}n < j \leq n - \bar{d}n}$ where a, b, c, d belong in the interval $(0, 1)$ and $\bar{a} = 1 - a$,

$\bar{b} = 1 - b$, $\bar{c} = 1 - c$, $\bar{d} = 1 - d$ $(\bar{a} + \bar{b} < 1, \bar{c} + \bar{d} < 1, p = round((1 - (\bar{a} + \bar{b}))m), q = round((1 - (\bar{c} + \bar{d}))n)$; in other words, the first $\bar{a}m$ rows, the last $\bar{b}m$ rows, the first $\bar{c}n$ columns, the last $\bar{d}n$ columns are removed from A.[2]

For composed reductions, we represent the combination of these reductions as a vector $\vec{\theta} = (k, s, a, b, c, d)$, where k is the quantization parameter, s the downsampling parameter, and a, b, c, d are the slice parameters in four directions.

4.1.2. Image Reductions

The reductions discussed in Section 4.1 are defined for an arbitrary matrix. Here we provide an overview of the quality reductions, implemented specifically over an image I of dimensions $C \times H \times W$ producing an image I' with the same dimensions in a deterministic way, having a reduction vector $\vec{\alpha}$ (input reduction value) whose dimension depends on the reduction. We will use $\vec{\alpha}$ to compute a parameter vector $\vec{\theta}$, which provides the parameters for reducing the image. We define atomic and composed reductions and provide examples for the given reductions.

4.1.2.1. Atomic Reductions

The following reductions are considered atomic reductions because they are applied to an image in only one dimension per step or with the same parameter per dimension, thus one parameter α will be applied.

4.1.2.2. Quantization

This quality reduction establishes the amount of colors that a pixel can take in RGB scale. Given an image I, with 255^3 available intensities per channel, and a reduction factor $\alpha \in (0, 1]$, $I_{\downarrow Q(\alpha)}$ leaves $round(255k)$ intensities per channel. This restriction reduces the possible colors of a given image to a total of $(round(255\alpha))^3$. Its mapped parameter vector is $\vec{\theta} = (\alpha, 1, 1, 1, 1, 1)$.

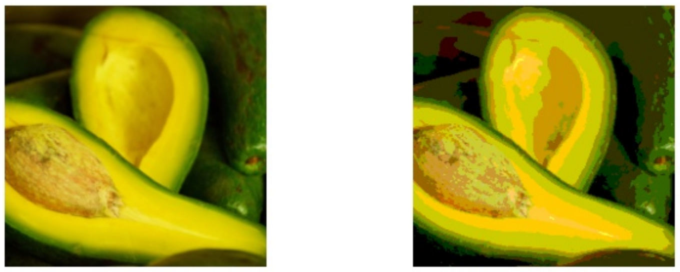


Figure 4.1: Application of Quantization Reduction with a α factor of 0.02. Source ‘Pixabay’.

[2] We use $\bar{a}$, $\bar{b}$, $\bar{c}$ and $\bar{d}$, to follow the intuition that higher valued parameters keep more entropy, and lower values reduce entropy more. Thus when $a = 1$, no rows are removed from the top.

4.1.2.3. Downsampling

Downsampling reduces the resolution of the image by resizing the image to a factor of its original resolution without applying a process of anti-aliasing (thus keeping the original image features) and then scaling it back to its original size for keeping compatibility with the Neural Network input. The image resize is made using the *torchvision* built-in transformation *resize*. For a given image I and a scaling factor $\alpha \in (0, 1]$, the application of the quality reduction downsampling produces $I_{\downarrow D(\alpha)}$ with dimensions $\lfloor \alpha H \rfloor \times \lfloor \alpha W \rfloor$. Its mapped parameter vector is $\vec{\theta} = (1, \alpha, 1, 1, 1, 1)$.

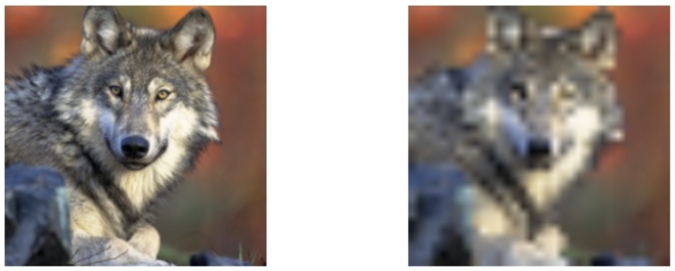


Figure 4.2: Application of Downsampling Reduction with an α factor of 0.15. Source 'Pixabay'.

4.1.2.4. Crop

This quality reduction applies a centered crop slice considering as a reduction factor the proportion of remaining area from the image, taking as reference the full image area; we denote the reduction of the area α as the remaining proportion area with respect to the full image, thus its range is in the interval $(0, 1]$. The output image has the same height and width as the input image and is an inner rectangle of unknown dimensions u, v with the same center as the image. Then $HW\alpha = uv$ and the horizontal and vertical distance p, q respectively between the rectangles gives us the relations $W = u + 2p$ and $H = v + 2q$. Also, considering the proportion of the cropped rectangle is the same as the original image we have $\frac{H}{W} = \frac{u}{v}$, then solving the equations for u and v we obtain the displacement values $p = \frac{V(1+\sqrt{\alpha})}{2}$ and $q = \frac{H(1+\sqrt{\alpha})}{2}$ from which we define the slice parameters for the reduction a, b, c, d as $\frac{1+\sqrt{\alpha}}{2}$. Then, its mapped parameter vector, obtained from the reduction parameter, is $\vec{\theta} = (1, 1, \frac{1+\sqrt{\alpha}}{2}, \frac{1+\sqrt{\alpha}}{2}, \frac{1+\sqrt{\alpha}}{2}, \frac{1+\sqrt{\alpha}}{2})$, and $\frac{1+\sqrt{\alpha}}{2} \in (0, 1]$. The cropped area is replaced by a gray value in RGB $((128, 128, 128))$[3].

[3] Gray value was used as the arithmetic mean between completely black and completely white for making reductions fairly, as the algorithm detected black as sea and white as lighted environment

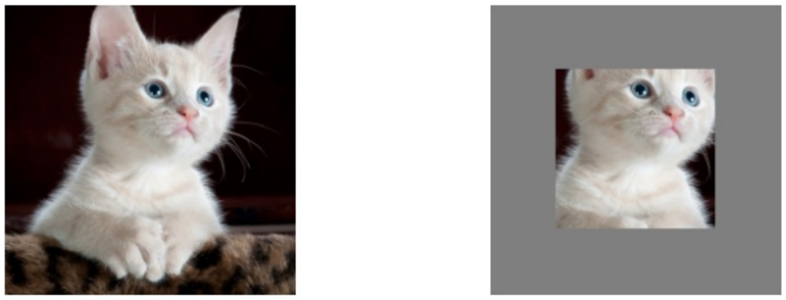


Figure 4.3: Application of Crop Reduction with a α factor of 0.30. Source 'Pixabay'.

4.1.2.5. Combined

This quality reduction applies the same reduction factor in all dimensions, considering for Slice reduction the percentage of the remaining area of the image as its parameter to make a center crop. This quality reduction methodology is used when a fixed reduction is applied. It receives as an input the reduction parameter α, which is then mapped to the following parameter vector $\vec{\theta} = (\alpha, \alpha, \frac{1+\sqrt{\alpha}}{2}, \frac{1+\sqrt{\alpha}}{2}, \frac{1+\sqrt{\alpha}}{2}, \frac{1+\sqrt{\alpha}}{2})$.

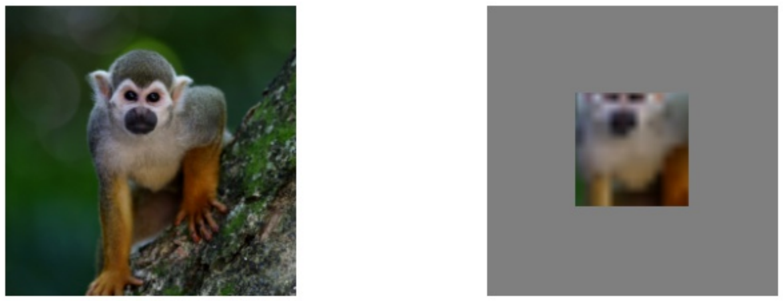


Figure 4.4: Combined Reduction Atomic Reduction Application with parameter $\alpha = 0.15$. Source 'Pixabay'.

4.1.3. Composed Reductions

The following reductions are considered composed reductions because they can be applied to an image with independent parameters in each dimension. In these reductions we consider a reduction vector $\vec{\alpha}$ which is then mapped to a parameter vector $\vec{\theta}$.

4.1.3.1. Slice

This quality reduction has four types of applications depending on the direction of the slice: from top to bottom and bottom to top are applied vertically (over rows) and left to right and right to left are applied horizontally (over columns). Each one replaces the values of the tensor, depending on its direction, to a gray value in the three channels of the sliced pixels. Its reduction vector $\vec{\alpha} = (a, b, c, d)$ is mapped to $\vec{\theta} = (1, 1, a, b, c, d)$.

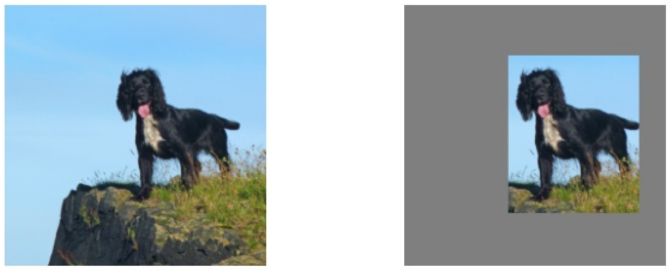


Figure 4.5: Application of Slice Reduction with parameters $a = 0.80, b = 0.80, c = 0.60, d = 0.90$. Source 'Pixabay'.

4.1.3.2. Combined

This quality reduction applies the reduction factor independently, for each reduction dimension per reduction step. Its reduction vector $\vec{\alpha} = (k, s, a, b, c, d)$ is mapped to the same parameter vector $\vec{\theta}$.

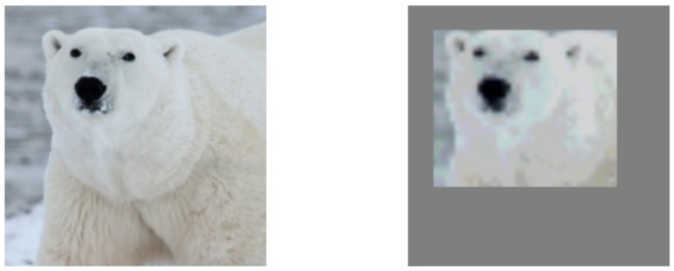


Figure 4.6: Application of Composed Combined Reduction with parameters $k = 0.05, s = 0.25, a = 0.90, b = 0.70, c = 0.90, d = 0.80$. Source 'Pixabay'.

4.2. Training Methodologies

The proposed training methodology pipeline is separated in distinct phases, which adds an *Image Reducer* module to the conventional training methodology; this module can receive feedback from the Neural Network classifier though a *Controller Module*, resulting in an architecture that changes depending on the current state of the training process. This methodology involves training the Neural Network with reduced images, simulating images with low quality in the described dimensions (i.e., Slice, Quantization and Downsampling) for achieving, according to our hypothesis, better performance over images with low quality or with loss of information. With this objective, we develop a module, which reduces the training image with this low quality condition, receiving feedback from the training process or the Neural Network depending on the type of methodology applied.

In this section we describe and justify the specific training methodologies used in this work in an incremental way; differences between methodologies depend on how they receive feedback from the Neural Network for obtaining the reduced image.

4.2.1. Standard Methodology

The Standard Methodology trains with images of full quality (without applying reductions to the images) during training. It passes the batch of the images and its labels directly (denoted as $(image\_batch, label\_batch)$) to the Neural Network in training mode and the metrics are processed in the *Controller* Module to check whether to stop or continue the current training, depending on the *patience* parameter that indicates the number of epochs to wait for a performance improvement, applying an early-stopping strategy to reduce the number of epochs necessary for training (this strategy is described in Section 2.6.2.2).

4.2.2. Fixed Reduction Methodology

This methodology trains with images using a quality reduction with a reduction vector $\vec{\alpha}$ that does not change between epochs, with the objective to build a quality-reduced image batch. This is achieved by passing a batch with its labels to the *Image Reducer* module, which reduces the images in the batch according to a reduction vector $\vec{\alpha}$, producing a reduced batch (denoted as $image\_batch'$) which is then passed to the Neural Network to train with. Note that the reduction vector $\vec{\alpha}$ also indicates whether or not to reduce the image (no reductions are applied if their values are 1).

The fixed methodology thus generalizes the standard methodology where the reductions are applied once over the training images in a preprocessing step, and is intended primarily as a baseline. The training stops when the *Controller* Module detects there are no increments in the accuracy of the validation set in a given *patience* number of epochs.

4.2.3. Linear Reduction Methodology

Since using a fixed value of the reduction vector $\vec{\alpha}$ for all epochs may oversimplify the training by using a single batch of images with the same level of reduction at each phase, we proposed to linearly adjust the values of $\vec{\alpha}$ between epochs per a *step* hyperparameter, with the objective of generating images with different levels of quality reductions. We expect this methodology may improve the performance of the Neural Network being trained since it receives different images in the process, ranging from full quality to low quality images. Thus the *Linear Reduction Methodology* trains with images with an atomic quality reduction, and with a single α parameter provided by the *Controller* Module that may change between epochs.

This methodology uses feedback from the validation set measure given by the Neural Network to adjust the current value of α, which depends on the current training measures and is adjusted by the *Controller* Module. The *Controller* Module also allows us to decide the amount of rounds and epochs for which a current value of α should be applied, and therefore allows us to test different variants on how to apply this reduction factor; we have considered alternating the value of $\vec{\alpha}$ between its current value and the value of one (full quality images) to mix the classic and reduced epochs (i.e., training over full-quality and then reduced images), and to use the same value until, in a *patience* number of epochs, the performance does not improve, finishing a round with a classic epoch; these training variants will be explained in Section 4.3. A diagram of this Training Methodology can be seen in Figure 4.7.

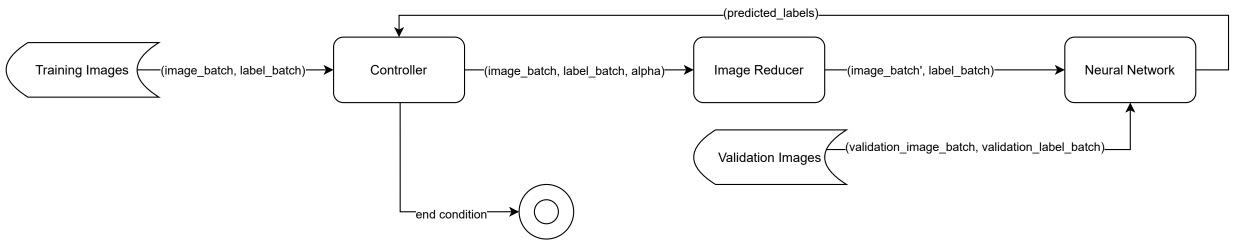

Training Images
(image_batch, label_batch)
Controller
(image_batch, label_batch, alpha)
Image Reducer
(image_batch', label_batch)
Neural Network
(predicted_labels)
Validation Images
(validation_image_batch, validation_label_batch)
end condition

Figure 4.7: Linear Reduction Methodology Pipeline.

4.2.4. Adaptive Reduction Methodology

Using the Linear Reduction Methodology, the same factor of reduction is applied over all dimensions. However, in the case of slice, we may rather wish, for example, to crop more from the left than the right if the object of interest is centered towards the right of the image. While we could linearly reduce the image in individual dimensions, or in combinations of dimensions, this would lead to a large search space, and also could end up training the Neural Network with images that do not actually depict the object that is labeled. We thus consider a methodology where we can guide the application of the quality reduction in each dimension separately by receiving direct feedback from the Neural Network.

The *Adaptive Reduction Methodology* precomputes a reduction vector $\vec{\alpha}$ using feedback from the Neural Network from the previous epoch in the *Parameter Generator* module. This is done by switching the Neural Network to evaluation mode and classifying the batch of images with the applied reduction, which depends on the value of *generate* that also indicates to the *Parameter Generator* module whether or not to update or fix the obtained reduction vector; if the classification is correct, it continues reducing per a *step* hyperparameter, otherwise it stops reducing. The goal is to produce a training image reduced to the threshold of the current performance of the Neural Network, which is received by the *Controller* module with a metrics parameter, while also avoiding reductions that completely remove information about the labeled object from the image (e.g., slicing too far in a particular direction). Unlike in the Linear Reduction Methodology, the individual values for $\vec{\alpha}$ may be different for different dimensions.

With the obtained parameters vector we pass them to the *Image Reducer* module with the *image_batch* to reduce them according to the reduction parameters obtained from the last round of Neural Network feedback. A diagram of this methodology can be seen in Figure 4.9. We describe the generation of the reduction vector in the Code 4.1[4].

Source Code 4.1: Adaptive Reduction Parameter Generator

```
def get_reduction_parameters(reduction_dimensions: list(bool), step: float):
    # Initialize mask of reduction applications
    m_alpha = [0,0,0,0,0,0]

    # Neural Network in Evaluation Mode
    net.eval()

    # While the tensor is not null
    while image not null:
```

[4] The pseudocode shows how the parameters are obtained for one image; in a batch it is applied for each image.

```
        # Check each applicable reduction dimension
        for r in range(where(reduction_dimensions)[1]):
            # Copies the current reduction vector
            current_m_alpha = m_alpha.clone()
            # If is allowed to reduced in that dimension
            if reduced[r]:
                current_m_alpha[r]+=1
                reduced_image = reduce(image, ones(len(reduction_dimensions))-
    ↪ current_m_alpha*step)
                correct = net(reduced_image)
                # If the classification is correct
                if correct: m_alpha[r]+=1
                # Otherwise, the algorithm will no longer reduce in that dimension
                else: reduced[i] = False

    # Return the reduction parameters obtained
    return ones(len(reduction_dimensions))-m_alpha*step
```

As an example of an application of this reduction we provide Figure 4.8, which starts with a full-quality image over which each reduction is applied sequentially. For each correct classification of the Neural Network we preserve that reduction for the next iteration, combining all reductions for which a correct classification is received. If a classification is incorrect we do not apply the reduction and we do not continue in that reduction dimension in the next iteration. In the next iteration, we have a new candidate reduced image from which the next reductions are applied. When no dimension can continue being reduced we stop the algorithm and provide the reduction vector $\vec{\alpha}$ as the final reduced image achieved by the current state of the Neural Network.

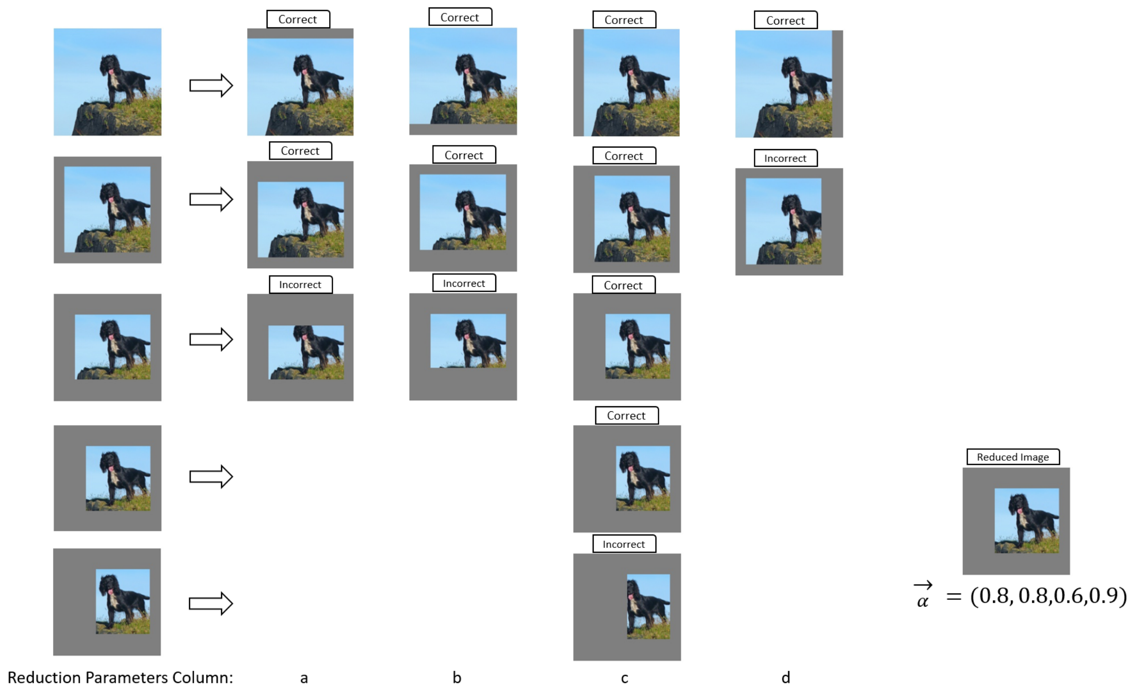


Figure 4.8: Application of Adaptive Reduction with *step* of 0.1 considering four Slice dimensions.

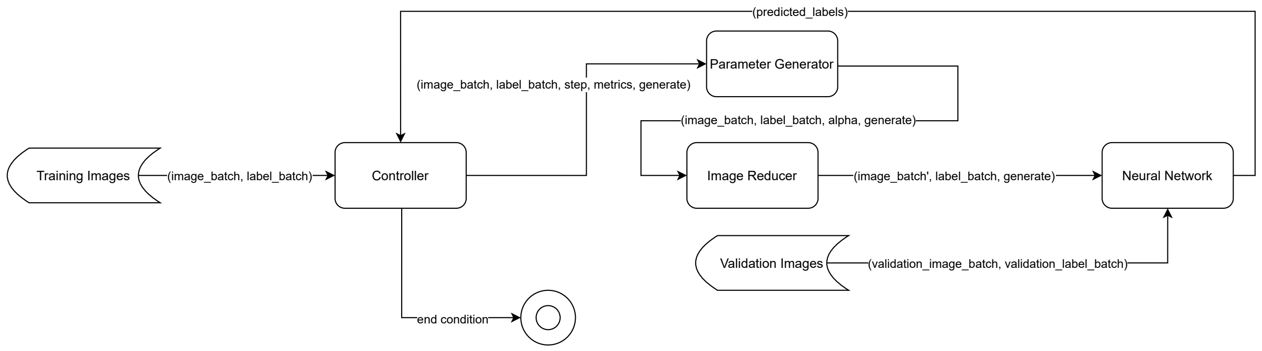

(predicted_labels)
Parameter Generator
(image_batch, label_batch, step, metrics, generate)
(image_batch, label_batch, alpha, generate)
Training Images
(image_batch, label_batch)
Controller
Image Reducer
(image_batch', label_batch, generate)
Neural Network
Validation Images
(validation_image_batch, validation_label_batch)
end condition

Figure 4.9: Adaptive Methodology Pipeline.

4.3. Linear and Adaptive Variants

In the Linear and Adaptive Reduction methodologies, the reduction parameters are changed between epochs. A natural idea is to start training with full quality images, and then train with increasingly reduced images. However, it is common in classical training methodologies to run several epochs for a given set of images. This leaves the question of when to change the reduction parameters. For example, after reducing images, we could run multiple training epochs with the same images, or we could run one epoch per reduction and apply this process several times. We now propose two variants of these methodologies that differ in how reduction parameters change over the epochs, which are now grouped into higher-level rounds.

4.3.1. Paired Epochs

In this variant, we apply rounds of one classic and one reduced epoch in which we use the current value of $\vec{\alpha}$, until the *patience* value is reached. This interleaving of classic and reduced epochs is inspired by preliminary experiments in which we found that classifier performance increases sharply after a classical epoch is applied with full quality images (we will see this behavior in Chapter 6). If the current accuracy of the validation set is surpassed we make a checkpoint. If the *patience* number is reached without hitting a checkpoint, we load the last checkpoint and continue with the next parameters of $\vec{\alpha}$ subtracting a *step* value from the current value. If after a *patience* number of rounds with different values of $\vec{\alpha}$ the validation accuracy did not improve, we stop the training.

This method skips the values of $\vec{\alpha}$ rounds that do not improve performance, leaving only the rounds that improved the accuracy of validation as part of the final training.

Source Code 4.2: Paired Epochs Algorithm

```
def paired_epochs(mask_reduction_dimensions: list(int), step: float, patience: int,
    ↪ best_checkpoint=None, validation_accuracy=0.0):

  # Initialization
  pairs_passed = 0
  rounds_passed = 0
  alpha_multiplier = 1.0
  performance_measures = PerformanceLog()

```

```
    while rounds_passed < patience:

        last_validation_accuracy = validation_accuracy
        alpha_multiplier = alpha_multiplier - step
        alpha = mask_reduction_dimensions*alpha_multiplier

        while pairs_passed < patience:
            if pairs_passed == patience: break
            # Reduced epoch
            reduced_state = net.train(alpha)
            # Classic epoch
            classic_state = net.train(ones(len(alpha)))
            # Update performance measures
            performance_measures.update(reduced_state.get_performance_measures(), classic_state.get_performance_measures())
            # Obtain best results of the pair
            current_validation_accuracy = max(current_reduced_validation_accuracy, current_classic_validation_accuracy)
            best_state = get_best_state(current_validation_accuracy)
            pairs_passed += 1
            # Check if validation accuracy is improved
            if current_validation_accuracy > validation_accuracy:
                validation_accuracy = current_validation_accuracy
                best_checkpoint = save_checkpoint(best_state)
                pairs_passed = 0
        rounds_passed += 1
        if last_validation_accuracy != validation_accuracy: rounds_passed = 0
        else: load_checkpoint(net, best_checkpoint)

    return performance_measures
```

4.3.2. Paired Rounds

In this methodology we set two *patience* parameters for the *Early Stopping* sub-module; the first is local with respect to the round's validation accuracy and the second is global with respect to the training process validation accuracy. As we are doing two *early stopping* processes in this methodology we proceed to distinguish them as *local* and *global patience* parameters respectively (in this work, they are set to the same value).

We apply reduced epochs with the same value of $\vec{\alpha}$ until the *local patience* parameter is reached. Then we do classic epochs until the same amount of *local patience* is reached. This ends a single round, and the training continues with the next value of $\vec{\alpha}$. If in any moment of the training the value of global validation accuracy is greater than the local validation accuracy, we make a checkpoint and we reset the global counter of rounds without improvement. If after a round the validation accuracy does not improve we load the last checkpoint and learning rate to skip the current value of $\vec{\alpha}$, and update it at the start of the next cycle by subtracting *step*. If after *global patience* rounds the global value of the validation accuracy did not improve we stop the training.

Source Code 4.3: Paired Rounds Algorithm

```python
def paired_rounds(mask_reduction_dimensions: list(int), step: float, patience: int,
    ↪ best_checkpoint=None, validation_accuracy=0.0):

    # Initialization
    rounds_passed = 0
    classic_epochs_passed = 0
    reduced_epochs_passed = 0
    alpha_multiplier = 1.0
    performance_measures = PerformanceLog()

    while rounds_passed < patience:

        last_validation_accuracy = validation_accuracy
        alpha_multiplier = alpha_multiplier - step
        alpha = mask_reduction_dimensions*alpha_multiplier
        local_validation_accuracy = 0.0

        while reduced_epochs_passed < patience:
            reduced_state = net.train(alpha)
            performance_measures.update(reduced_state.get_performance_measures())
            reduced_epochs_passed += 1
            if reduced_state.validation_accuracy() > local_validation_accuracy:
                local_validation_accuracy = reduced_state.validation_accuracy()
                reduced_epochs_passed = 0
            if reduced_state.validation_accuracy() > validation_accuracy:
                rounds_passed = 0
                reduced_epochs_passed = 0
                validation_accuracy = reduced_state.validation_accuracy()
                best_checkpoint = save_checkpoint(reduced_state)

        while classic_epochs_passed < patience:
            classic_state = net.train(alpha)
            performance_measures.update(classic_state.get_performance_measures())
            classic_epochs_passed += 1
            if classic_state.validation_accuracy() > local_validation_accuracy:
                local_validation_accuracy = classic_state.validation_accuracy()
                classic_epochs_passed = 0
            if classic_state.validation_accuracy() > validation_accuracy:
                rounds_passed = 0
                classic_epochs_passed = 0
                validation_accuracy = classic_state.validation_accuracy()
                best_checkpoint = save_checkpoint(classic_state)

        # Performance did not improve
        if last_validation_accuracy == validation_accuracy:
            rounds_passed += 1
            load_checkpoint(net, best_checkpoint)
        # Reset counter
        else: rounds_passed = 0

    return performance_measures
```

Chapter 5

Experimental Design

In this chapter we detail the research questions and the experimental design of this work. The architecture's implementation, experiments and dataset mapping can be found in our GitHub repository.[5].

5.1. Training with Quality-Reduced Examples

The task of Image Classification has improved in recent years with new convolutional architectures and methodologies over these architectures, but how to classify images in more challenging conditions (e.g., low resolution, a sliced image, presence of noise) remains an open problem in relation to the performance of these classifiers.

With the objective of addressing this problem, we study the performance of an approach that integrates classic epochs and reduced epochs during training for improving Image Classification in both standard conditions (over full-quality images) and challenging conditions (over reduced images). The research questions we propose in this work are the following:

Q1 Which training methodology is better in terms of the accuracy of Image Classification (considering both full-quality and reduced images)?

Q2 Do the proposed training methodologies using reduced images reduce the calculated entropy quotient (defined in Chapter 3.4) compared to the baseline of training (only) over full quality images?

5.2. Experimental Setting

In this section we describe the setting for the experiments in terms of hardware, datasets and Neural Networks used.

5.2.1. Hardware

The experiments were performed on a computer with an 8 core AMD Ryzen 7 1800X CPU, a *64* bit architecture, *94* GBs of RAM, and an Nvidia RTX3090 GPU with 24 GB of G6X memory. The operating system is Linux with the Arch distribution. The Neural Network models are provided by the Pytorch Python Library, version *1.9.0+cu111* and Nvidia CUDA version *11.5*.

[5] GitHub Repository: https://github.com/pabtorres/quality-reduced-training

5.2.2. Datasets

We use training, validation and test datasets based on ImageNet [21]. Given that ImageNet does not explicitly provide the labels of the testing dataset, a total of $25,000$ images were picked at random from the training set to be used as a validation set and the provided validation dataset was used as a test dataset, as is often done in the literature [6].

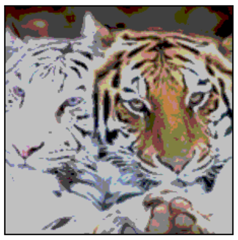
Given the prohibitive costs of training models over the full ImageNet dataset, we also define a smaller subset on which we run initial experiments over numerous configurations. We build a "HumaNet" subset from ImageNet restricted to the 20 classes defined by Carrasco et al. [9]. To have a balanced number of images per class, which reside at different levels of the ImageNet hierarchy, we apply subsampling of these images, with respect to the number of images in the class with fewest images, distributing the super-classes with more classes evenly in the obtained sets for the training and validation sets which both are full-quality images. Using the same classes as HumaNet further allows us to use the test set of reduced images computed by human users, made available by Carrasco et al. [9] as we now describe.

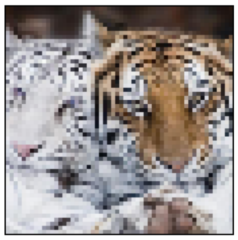
Table 5.1: Datasets description.

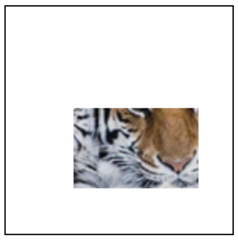
Dataset	**Training Images**	**Validation Images**	**Test Images**
ImageNet	$1,281,167$	$25,000$	$50,000$
HumaNet	$25,000$	500	1000

HumaNet also provides testing datasets with reduced images; these evaluation datasets are: Color, Combined, Crop and Resolution, each one having a total of 300 images and the Complete HumaNet dataset with 1000 images. These test sets were generated by Carrasco et al. [9] via a bottom-up approximation of a Human MEPI, starting with a void image and allowing a user to increase the image quality in each step until they can correctly guess the class of the image (in which case the image is included in the set) or they guess incorrectly (in which case the image is excluded and a new void image is loaded). We also provide a HumaNet test based on the ImageNet test set with 50 images per class. In Figure 5.1 we provide an example of the different reduction dimensions of the MEPIs obtained from human users.

(a) Original


(b) Color


(c) Resolution


(d) Crop

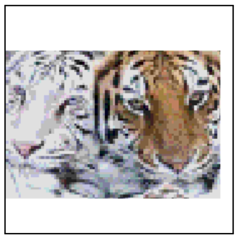

(e) Combined

Figure 5.1: Example of the HumaNet test images with MEPIs computed by humans

5.2.3. Neural Networks

In this subsection we briefly describe and justify the Neural Networks chosen. Also we provide the hyper-parameters used, which were obtained from each Neural Network paper, and how we adjusted the neural networks to classify 20 classes.

5.2.3.1. SqueezeNet

SqueezeNet is a Convolutional Neural Networks that achieves similar performance results as AlexNet, but with 2% of the number of parameters, facilitating the deployment of this classifier on devices with low memory. It was proposed at the end of 2016 by Iandola et al. [5].

5.2.3.2. ResNet

Residual Neural Network is a Deep Neural Network proposed in 2016 by He et al. [4], which achieved, in its ResNet-152 variant, state-of-the-art performance on ImageNet, as a model which did not use extra training data.

5.2.3.3. EfficientNet

EfficientNet is a Convolutional Deep Neural Network proposed in mid 2019 by Tan et al. [6], which achieved, in its EfficientNetB7 variant, state-of-the-art performance on ImageNet. This model has been used in different methodologies and variants, also achieving state-of-the-art performance on the ImageNet dataset even in 2021. As EfficientNetB7 consumes more resources than those available to us we used EfficientNetB3 in our experiments, which is a more lightweight version of EfficientNetB7.

5.2.3.4. Network Hyperparameters

In Table 5.2 we provide the hyperparameters based on those proposed by the authors of each Neural Network and availability in the Pytorch library. For EfficientNet in ImageNet we used a learning rate of 0.00001 to avoid divergence of the loss function.

Table 5.2: Neural Networks hyperparameters. LR: Learning Rate, WD: Weight Decay

Model	Optimizer	LR	Momentum	WD	Scheduler	Step Size	Gamma
ResNet	SGD	0.1000	0.9	0.00010	StepLR	10	0.10
SqueezeNet	SGD	0.0010	0.9	0.00020	StepLR	10	0.10
EfficientNet	RMSProp	0.0001	0.9	0.00001	StepLR	3	0.97

5.2.3.5. Adjustment

The aforementioned Neural Networks assume 1000 classes. For the purposes of HumaNet, with 20 classes, in every Neural Network we added a fully connected layer, from its original 1000 classes to 20, reducing the available labels to the ones studied by Carrasco et al. [9].

5.3. Training Methodologies Tested

We propose a series of incremental experiments to answer our research questions starting from a baseline with no reductions, and then experimenting with a fixed reduction in all epochs. Based on the obtained results we propose the experiments mentioned in Section 4.3 in which we combine classic and reduced epochs.

Because of the size of the full dataset of ImageNet and the cost of training models for it, a baseline and one methodology (the best overall) will be tested based on the results for the HumaNet subset.

5.3.1. Exploratory Analysis

Initially we compared the performance of training over full-quality images versus reduced-quality images seeing a peek in performance for epochs training with full-quality images that directly followed various epochs training with reduced-quality images (we will see this behavior in more detail in Chapter 6); thus, in the final training methodologies tested, we include alternatives that interleave full-quality and reduced-quality epochs.

We also tested with fixed values of $\vec{\alpha}$ across training epochs. We then add a *step* value that adjusts $\vec{\alpha}$ at the start of the cycle, where the Neural Network training discards values that do not improve performance. The smaller the value of *step*, the more fine-grained the adjustments, but the higher the cost of training given that reductions are applied more slowly.

5.3.2. Start Point

In order to have a start point for our methodologies, we ran classic epochs with the same *patience* value of 3 for the *early stopping* process over the *validation* dataset for each Neural Network (for more information about the *patience* hyperparameter, we refer to Section 2.6.2.2).

5.3.3. Baseline

As the number of epochs needed for each methodology to terminate in each Neural Network model would be different, we established a baseline with the maximum number of epochs achieved by each model in our methodology. The baseline thus uses at least as many epochs as the methodologies to which we compare.

5.3.4. Fixed Reductions

We start using only one reduction over all images with a fixed value of $\vec{\alpha}$ for all reduction dimensions and epochs. We continue training until the *patience* parameter is reached in the *early stopping* process. This methodology is equivalent to the baseline, but where images are reduced once prior to training.

5.3.5. Linear Reductions

All other methodologies take the model produced by the starting point. We use a *step* parameter of 0.125 that will update the reduction vector and a *patience* parameter of 3. We tested our methodology with the paired epochs and paired rounds variants.

5.3.6. Adaptive Reductions

We again consider the best checkpoint of the baseline as our starting point. We use a *step* parameter of 0.125 that will be used in the generation of the reduced images and a *patience* parameter of 3. We tested our methodology with the paired epochs and paired rounds variants.

5.4. MEPIs Calculation

To calculate the entropy reduction ratio, we computed the Minimal Entropy Positive Images appling the same process as Carrasco et al. [9] to minimize the size in bytes of the images using quality reductions defined in Section 4.1.

The process of obtaining an exact *Minimal Entropy Positive Image* of a multi-variable reduction (i.e., slice and combined) using a brute force algorithm has a high algorithmic cost, thus Carrasco et al. [9] proposed to use Powell's Algorithm for optimizing the application of the reductions subject to minimizing the entropy measured as the size of the image as a lossless compression, particularly the PNG format, searching for the *Monotonic Minimal Entropy Positive Image* because of the monotonic behavior of the application of the quality reductions. As a starting point that accelerates the search we considered the following reduction vector $\vec{\alpha} = (k, s, a, b, c, d) = (0.1, 0.95, 0.925, 0.925, 0.925, 0.925)$. As greater downsampling and slice reductions produce more impact in the image pixels we considered a lower reduction on those dimensions as a starting point. Note that Powell's Algorithm can search by increasing or decreasing parameters, so if the full image is classified correctly, but the image with the starting point reduction is not, the search algorithm can also increase the parameters to improve the quality of the image.

Chapter 6

Results

In this chapter we provide the results obtained for the experiments for both datasets, and then we discuss these results, answering the research questions of this work.

6.1. HumaNet

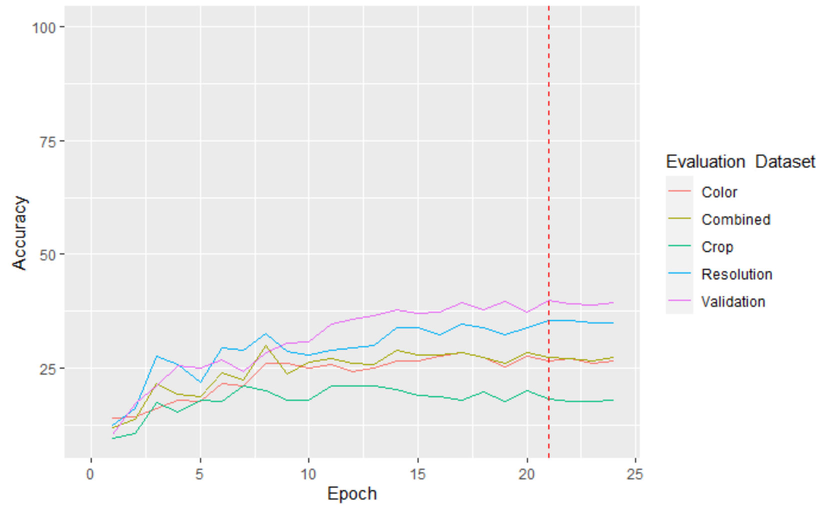
In this section we provide the results obtained on the HumaNet dataset.

6.1.1. Start Point

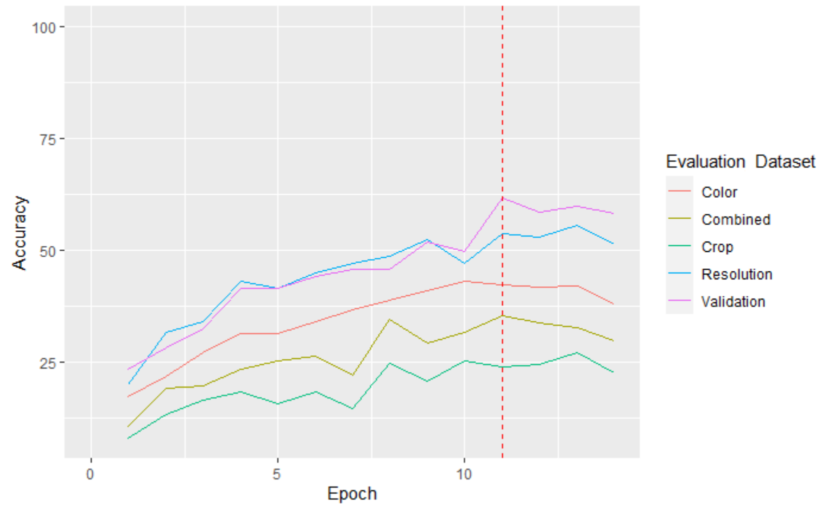
Table 6.1 features the results achieved for the start point of each Neural Network in each evaluation dataset in terms of accuracy. The start point involves running classical epochs of training over the full quality images until the patience parameter (which is 3) is reached through early stopping. These results represent the best validation accuracy and we also provide the epoch in which that result was obtained. The results in the datasets Color, Combined, Crop, Resolution and Test are the results obtained in the epoch that achieved the best validation accuracy (e.g. for EfficientNet-B3 the shown Color accuracy was obtained in epoch 19). In Figure 6.1 we show the evolution of the accuracy in the aforementioned datasets and the best validation accuracy is highlighted with a vertical line to also show the obtained performance on those epochs in the other datasets.

Table 6.1: Start Point results for accuracy.

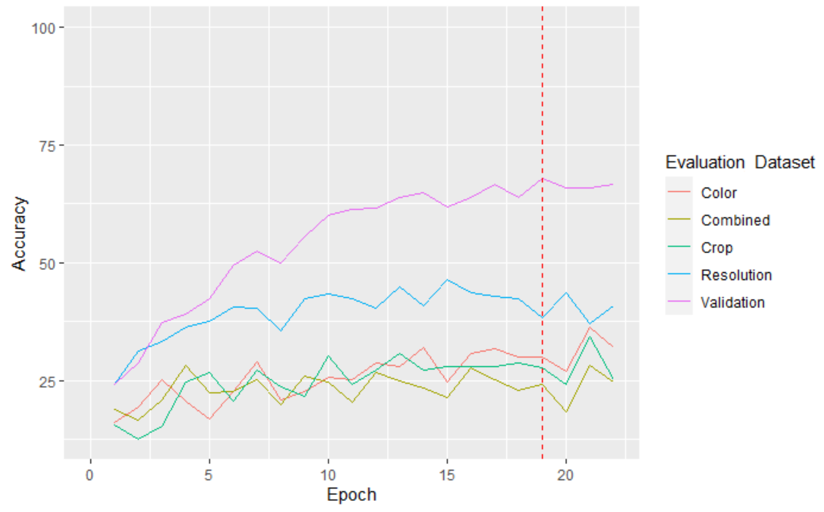
Model	Validation	Color	Combined	Crop	Resolution	Test	Achieved Epoch
SqueezeNet	39.8	26.6	27.3	18.3	35.3	39.0	21
ResNet-152	61.6	42.3	35.3	24.0	53.6	60.4	11
EfficientNet-B3	68.0	30.0	24.3	27.6	38.3	69.7	19



(a) Start Point performance: SqueezeNet



(b) Start Point performance: ResNet-152



(c) Start Point performance: EfficientNet-B3

Figure 6.1: Start Point evolution of baseline

We can observe that the performance of the start point for a given dataset depends on the Neural Network. For example, the best results for Resolution are seen for ResNet-152, while the best results for Crop are seen for EfficientNet-B3. This suggests that different Neural Networks might be sensitive to different types of information. In the following sub-sections, these results will be used for comparing our training methodologies.

6.1.2. Fixed Reduction

In Figure 6.2 we show the obtained accuracy using the fixed reduction methodology for training in each evaluation dataset taking the model from the epoch which achieved the best validation accuracy for each α value used. The values of α used were from 0.125 to 1.000 in steps of 0.125. When $\alpha = 1.000$ the results correspond to the validation accuracy of the start point (i.e., training without reductions). In each column we provide the name of the applied quality reduction during training.

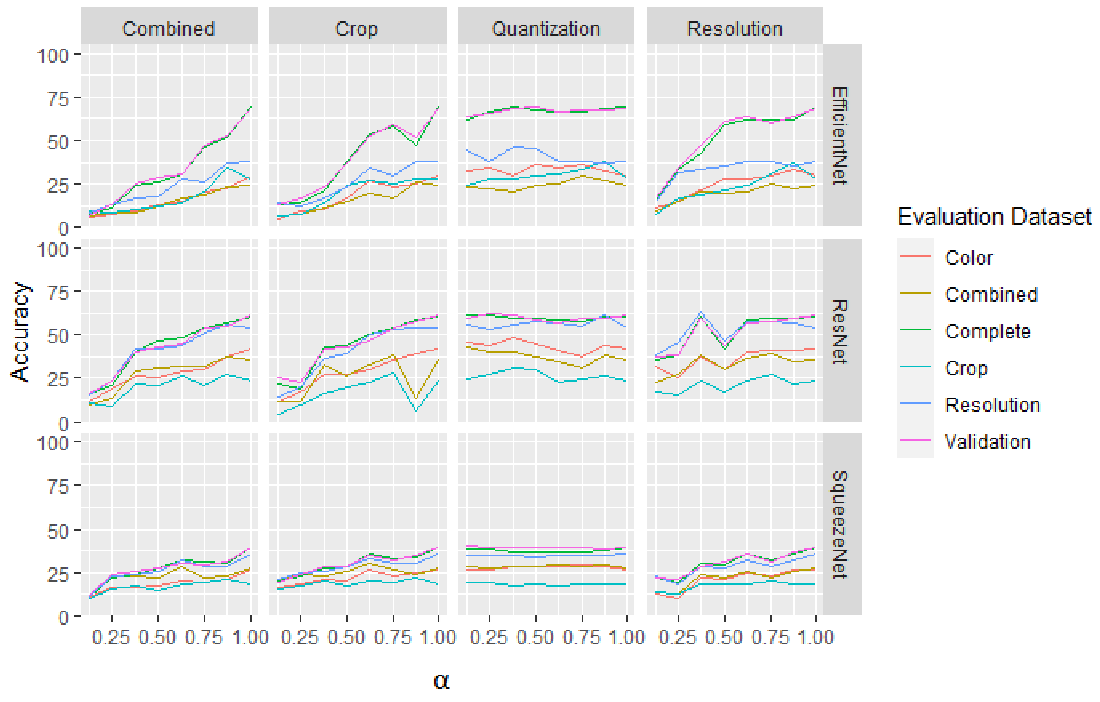


Figure 6.2: Obtained results for Fixed Reduction Experiment. The columns indicate the reduction dimension in which the Neural Network was trained.

In this experiment we can see a trend in the results in terms of the application of the reduction. If α is lower, performance decays; if α is higher (closer to a full-quality image) the performance is better. With these results we noted that training only with reduced epochs did not improve accuracy. These initial results inspired us to propose the linear and adaptive approaches that combine classic and reduced epochs.

Comparing the trend of the curves for each reduction type, we can observe that Quantization is less sensitive to the value of α. We can deduce from the definition of this reduction and the values used that the color reduction does not have a big impact on classification in the range $]0.375, 1.000]$ as can be seen in the curve of the Color evaluation dataset, even though images having 37.5% intensities per channel leave only 5.3% of the original colors at

$\alpha = 0.375$; as Carrasco et al. [9] observed, Neural Networks used for the Image Classification task are not as sensitive to color changes as they are to changes in other dimensions.

6.1.3. Linear Reduction

As a fixed value of α shows a drop in validation accuracy, we now present the results of experiments that change the value of α between epochs following a linear reduction. We provide the obtained results for the variants Paired Epochs and Paired Rounds.

6.1.3.1. Paired Epochs

As the reduced epochs lowered the performance, but the Neural Network has learned from the reduced images, we try to combine both classic epochs and reduced epochs for improving the performance of Image Classification. In this variant, we apply a classic epoch immediately after each reduced epoch. In Figures 6.3, 6.4, 6.5 we show the performance evolution of the evaluation datasets in terms of the accuracy of each Neural Network in every reduction dimension considering paired epochs. We also highlight with a vertical line the starting point of the reduced methodology (i.e., the point where the start point ends).

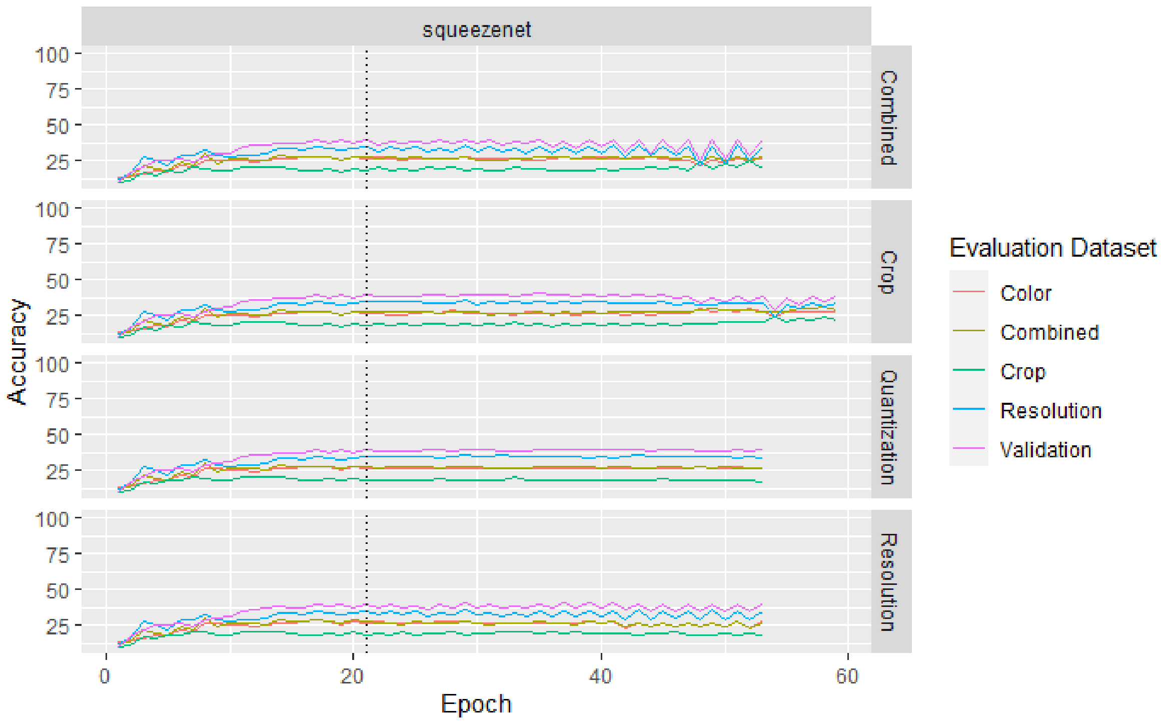


Figure 6.3: Performance evolution of SqueezeNet during training: HumaNet validation and test sets, linear reduction, paired epochs.

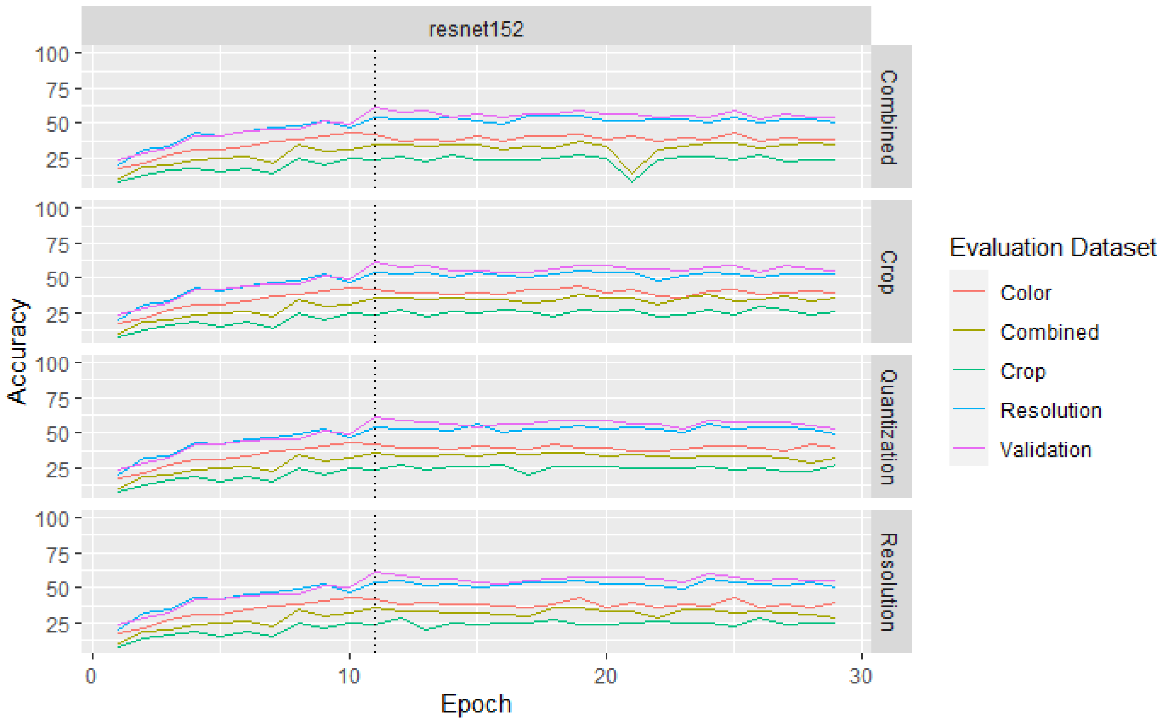


Figure 6.4: Performance evolution of ResNet during training: HumaNet validation and test sets, linear reduction, paired epochs.

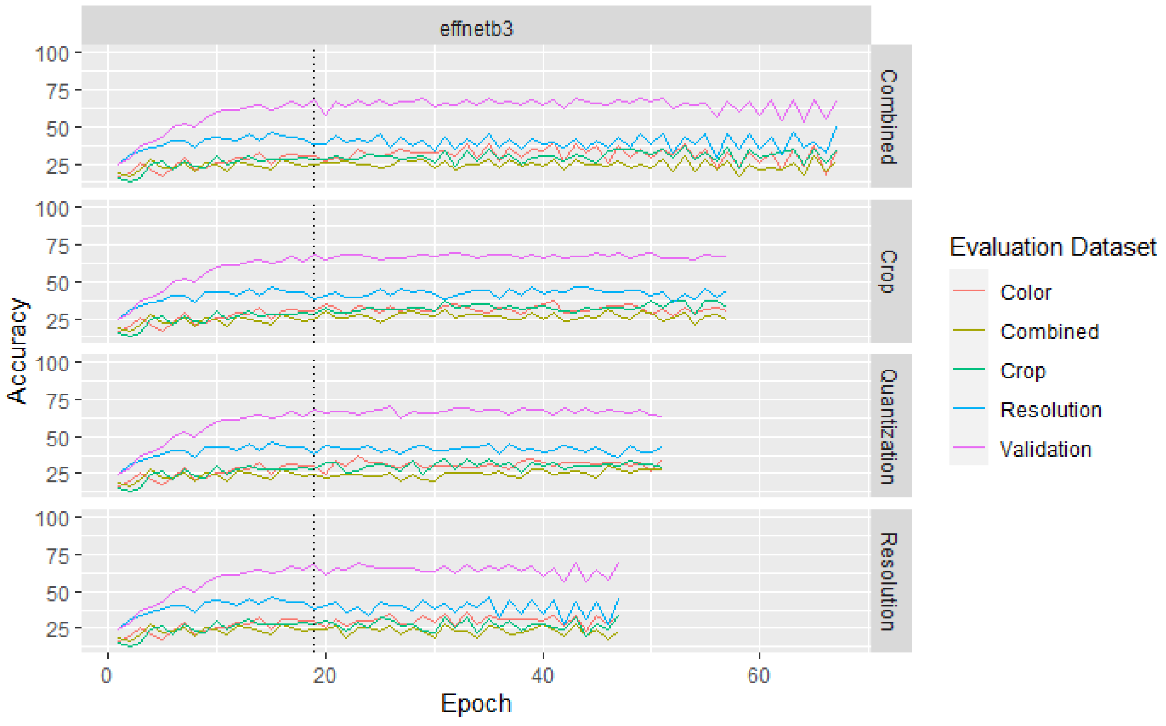


Figure 6.5: Performance evolution of EfficientNet during training: HumaNet validation and test sets, linear reduction, paired epochs.

From the obtained results we see that the methodology performs more epochs when the evaluation accuracy increases, and as an implication, more α values are tested. This implies that reductions closer to 0 are applied and it produces a drop in performance in the reduced epochs and an increase in the classic epochs producing a periodical "sawtooth" effect. After applying a reduced epoch, the classic epoch that follows sometimes manages to boost accuracy beyond the start point.

The results of SqueezeNet and EfficientNet-B3 improved with respect to the start point, but ResNet-152 did not, so different Neural Networks exhibit different performance using this methodology.

The best results obtained were by the Neural Network EfficientNet-B3 and reduction dimension quantization with a validation accuracy of 70.2%.

6.1.3.2. Paired Rounds

Considering that a sequence of a reduced epoch and a classic epoch may be restrictive, we consider that the Neural Network indicates, in terms of performance, the amount of epochs to apply of each type. Thus instead of pairing reduced and classic epochs, we will pair reduced and classic rounds of multiple epochs (the number of epochs in each round is decided by early stopping). In Figures 6.6, 6.7, 6.8 we show the performance evolution of the evaluation datasets in terms of the accuracy of each Neural Network in every reduction dimension. We also highlight with a vertical line the starting point of the reduced methodology (i.e., the point where the start point ends).

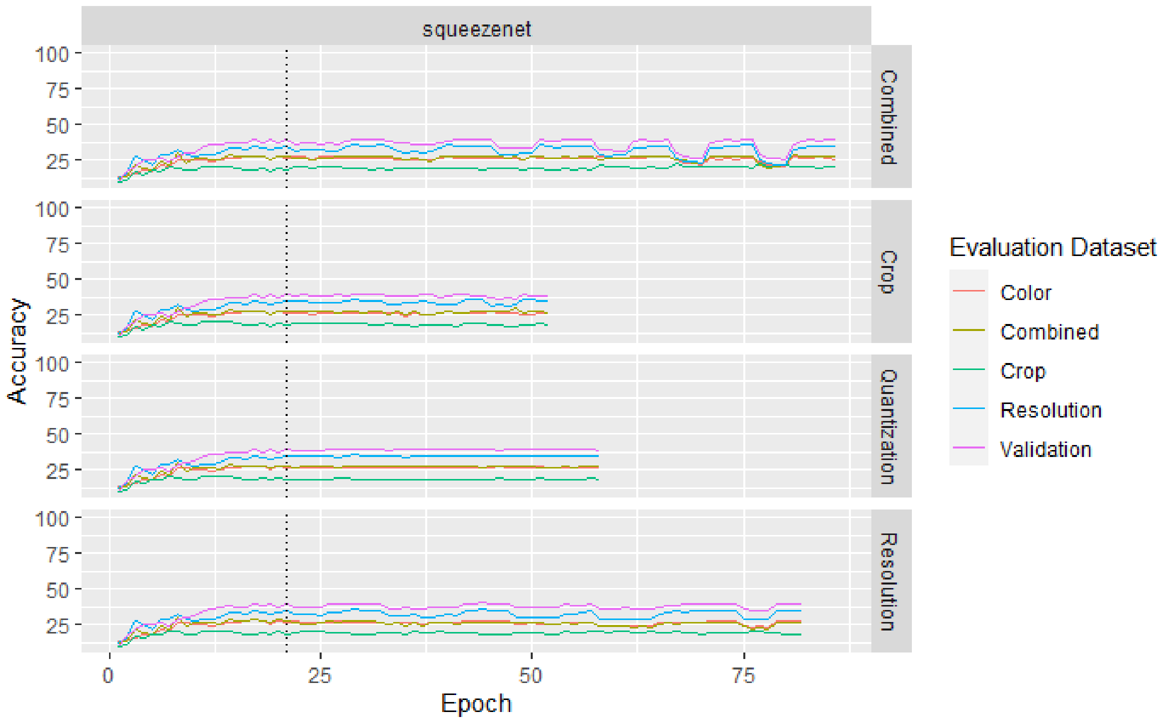


Figure 6.6: Performance evolution of SqueezeNet during training: HumaNet validation and test sets, linear reduction, paired rounds.

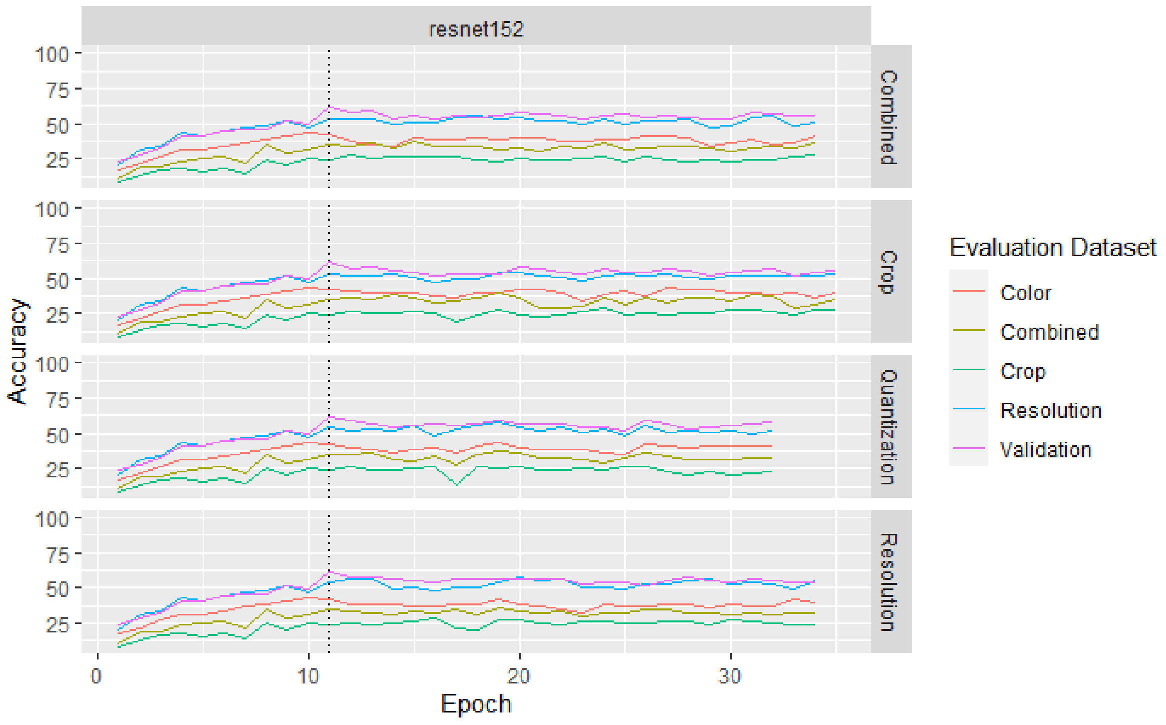


Figure 6.7: Performance evolution of ResNet during training: HumaNet validation and test sets, linear reduction, paired rounds.

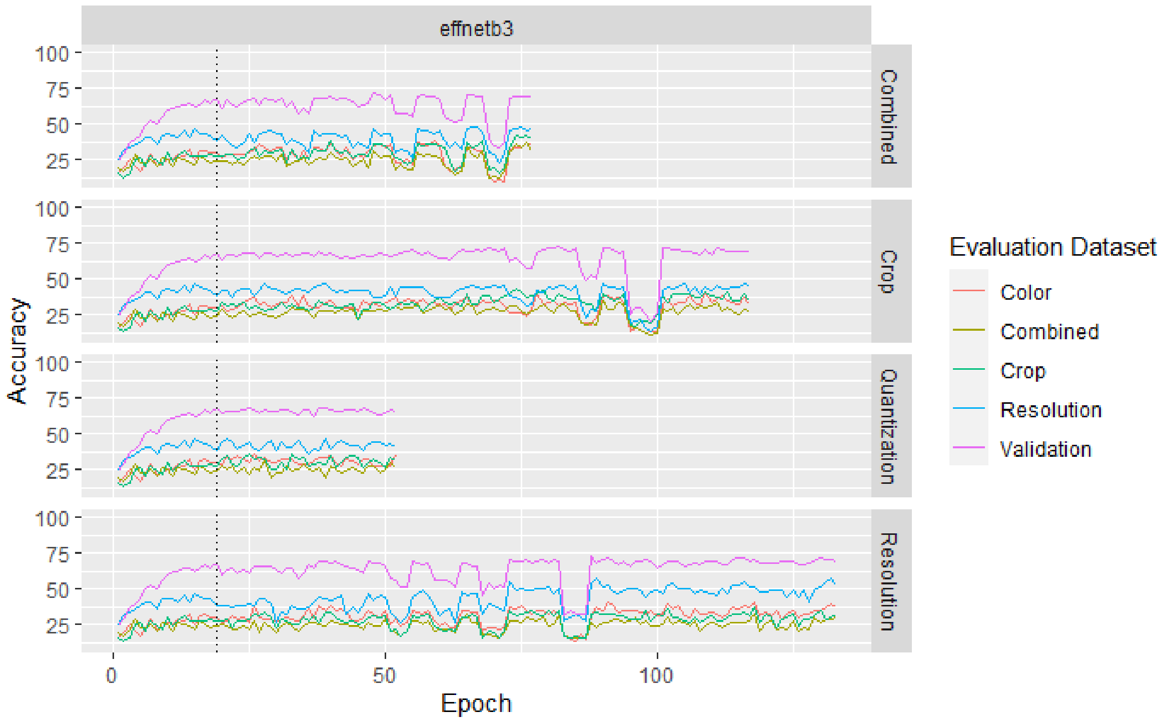


Figure 6.8: Performance evolution of EfficientNet during training: HumaNet validation and test sets, linear reduction, paired rounds.

From the obtained results we see that some reduction dimensions have more training epochs, thus more values of α to train with the reduced epochs. Also, from the amount of reduced epochs in every round, we observe a similar number of these epochs because of the decay of the performance. In terms of classic epochs the methodology can adapt the needed amount in each round so it varies in different rounds. This gives rise to a more square wave evolution in performance, where low plateaus correspond to a round of reduced epochs, and high plateaus correspond to a round of classic epochs. The magnitude sometimes increases as alpha decreases (leading to lower troughs). Again the classic epochs that follow a reduced round sometimes exceed the start point accuracy.

We can also observe that different reduction dimensions have different performance in terms of the epochs and values of α applied. In particular, we notice that ResNet is less sensitive to reduced epochs.

The best results obtained were by the Neural Network EfficientNet-B3 and reduction dimension crop and resolution with the same validation accuracy of 72.6%.

6.1.4. Adaptive Reduction

In our final methodology, rather than using a fixed or linearly decreasing value for α across all dimensions, we use an adaptive approach where the Neural Network from the previous epoch is used to decide the extent of the reduction in each dimension separately, creating a training image at the limit of the current Neural Network performance. We provide the result obtained for the variants Paired Epochs and Paired Rounds.

6.1.4.1. Paired Epochs

In Figures 6.9, 6.10, 6.11 we show the performance evolution of the evaluation datasets in terms of the accuracy of each Neural Network in every reduction dimension. We also highlight with a vertical line the starting point of the reduced methodology (i.e., the point where the start point ends).

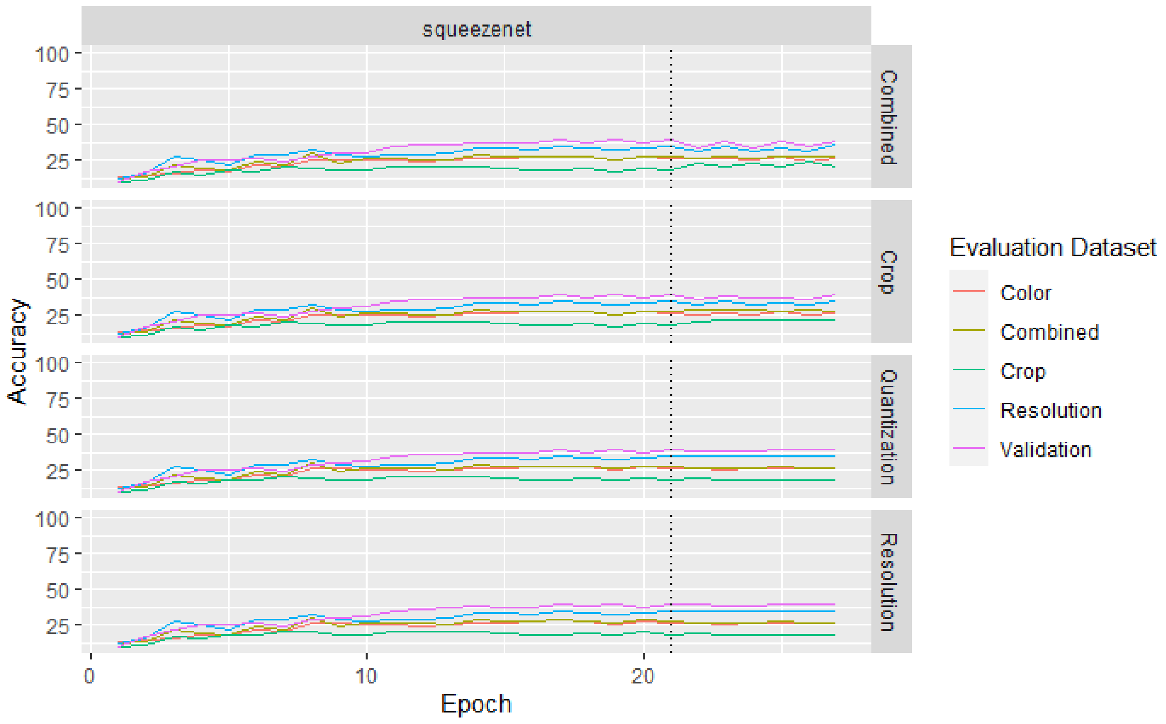


Figure 6.9: Performance evolution of SqueezeNet during training: HumaNet validation and test sets, adaptive reduction, paired epochs.

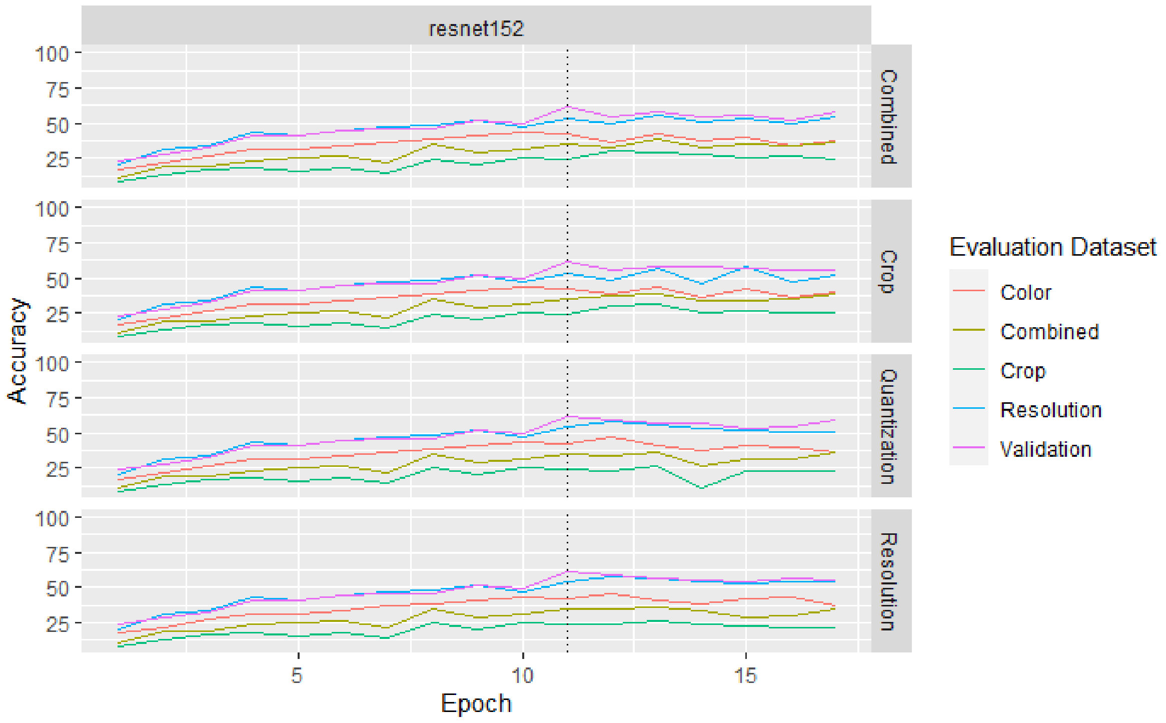


Figure 6.10: Performance evolution of ResNet during training: HumaNet validation and test sets, adaptive reduction, paired epochs.

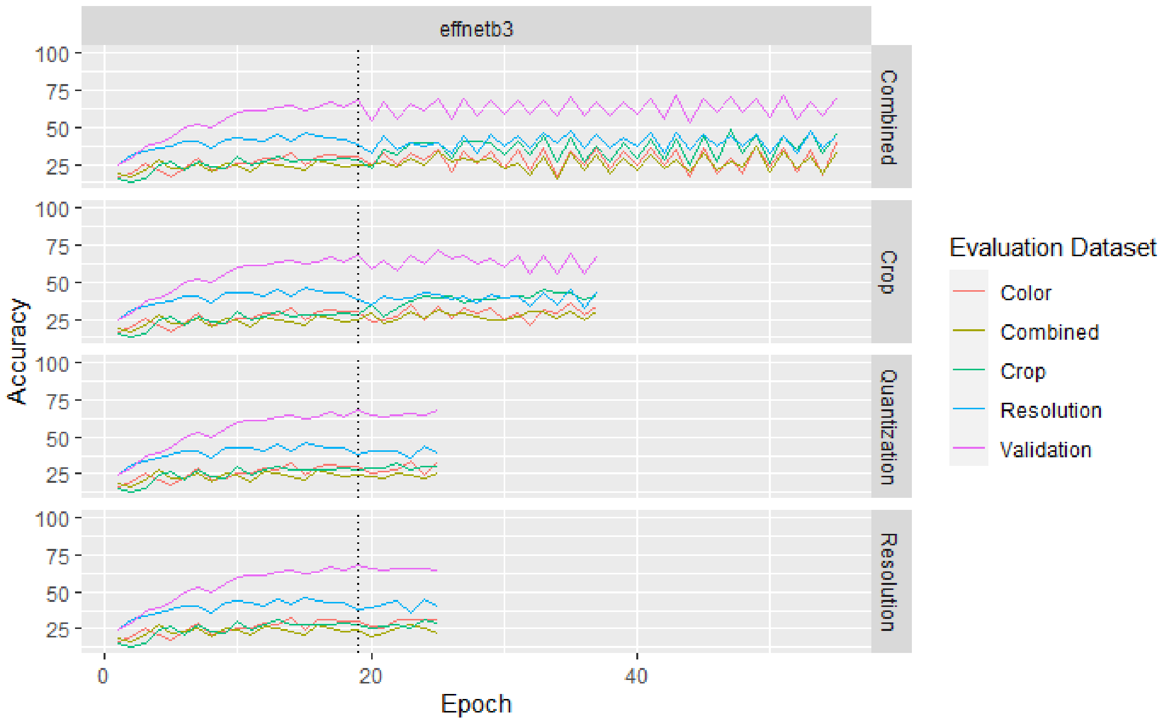


Figure 6.11: Performance evolution of EfficientNet during training: HumaNet validation and test sets, adaptive reduction, paired epochs.

From the obtained results, there is a difference in terms of the trained epochs depending on the quality reduction dimension (e.g. in the case of EfficientNet-B3 the combined reduction achieved more epochs). Notice that the other reductions in the adaptive methodology are a subset of the combined reduction dimension because it can express all the combinations from its reduction vector.

We can also see the periodical "sawtooth" effect whose amplitude is more stable, in terms of the increase and decrease of the performance, compared to the Linear methodology. We further see that the effects of the methodology vary between the different Neural Networks, with EfficientNet again showing more sensitivity to reduced epochs.

The best results obtained were by the Neural Network EfficientNet-B3 and reduction dimension combined with a validation accuracy of 71.4%.

6.1.4.2. Paired Rounds

In Figures 6.12, 6.13, 6.14 we show the performance evolution of the evaluation datasets in terms of the accuracy of each Neural Network in every reduction dimension again considering the adaptive methodology, but this time under the paired rounds variant. We also highlight with a vertical line the starting point of the reduced methodology (i.e., the point where the start point ends).

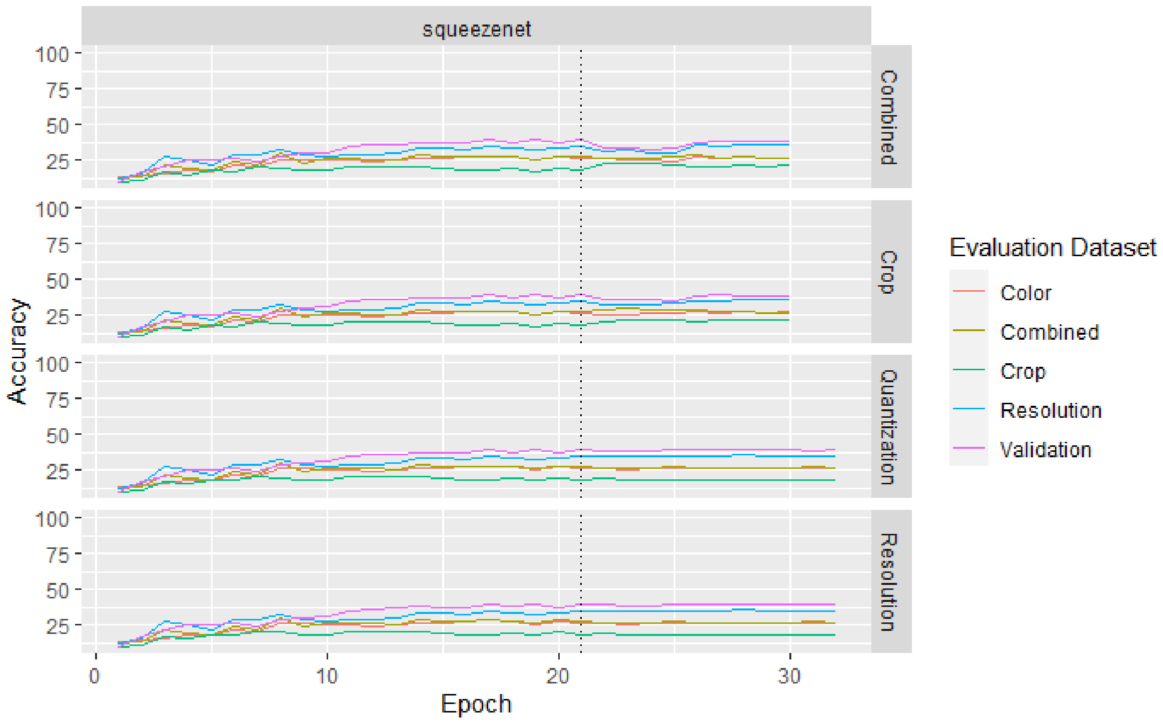


Figure 6.12: Performance evolution of SqueezeNet during training HumaNet: validation and test sets, adaptive reduction, paired rounds.

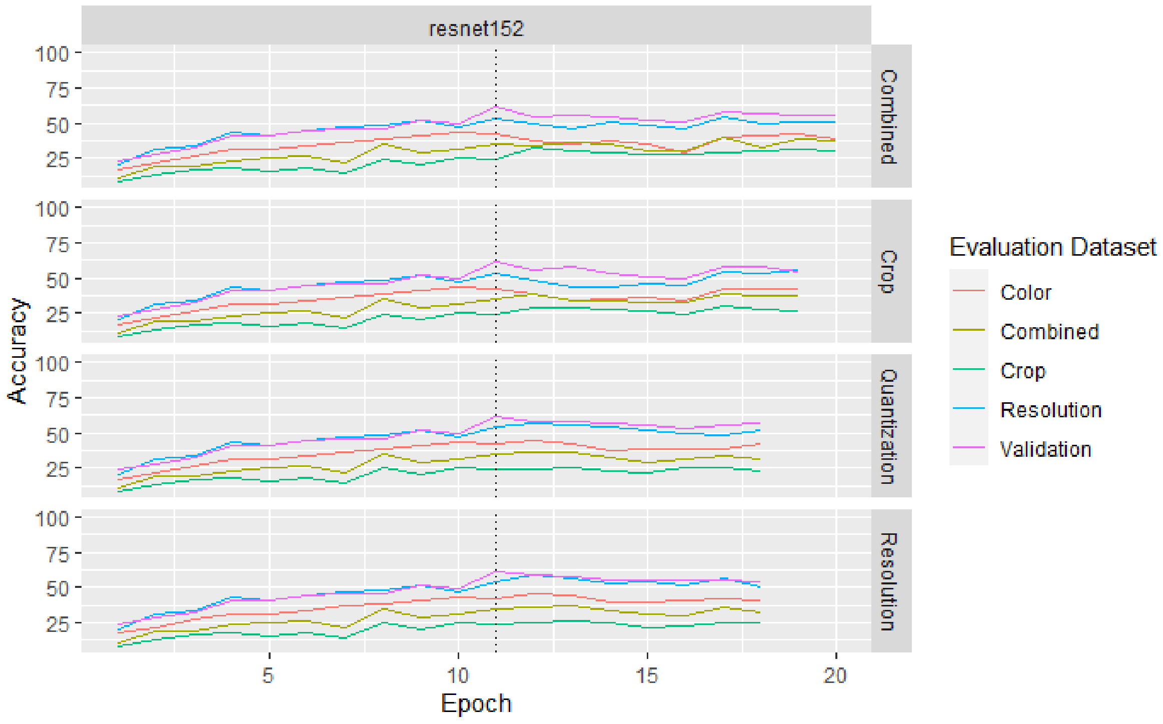


Figure 6.13: Performance evolution of ResNet during training HumaNet: validation and test sets, adaptive reduction, paired rounds.

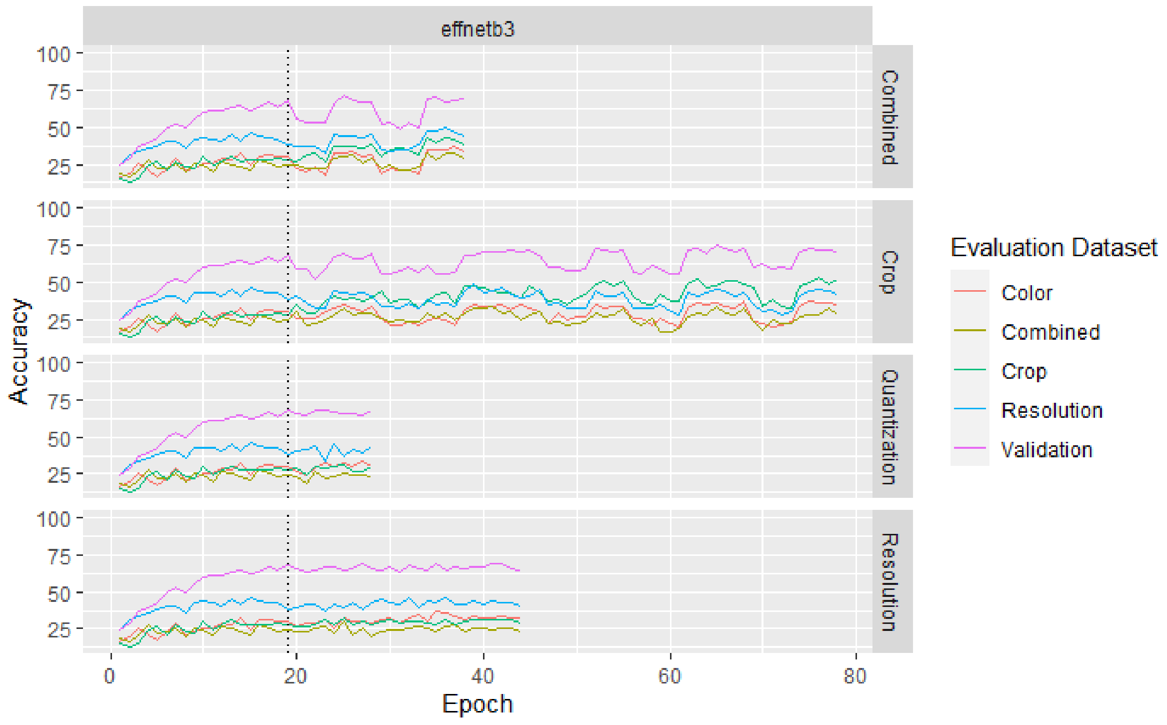


Figure 6.14: Performance evolution of EfficientNet during training HumaNet: validation and test sets, adaptive reduction, paired rounds.

From the obtained results we can observe a periodicity in terms of the drop of performance of the validation accuracy whose magnitude is more stable in terms of the increase and decrease of the performance. Also the best results obtained were by the Neural Network EfficientNet-B3 and reduction dimension crop with a validation accuracy of 75%.

6.1.5. Test Results

Here we provide a summary of the test results in a table considering the performance of each Neural Network in terms of the accuracy. Noting that methodologies involving reduced epochs apply many more epochs overall, we introduce a stronger baseline than the start point. Namely, for our baseline, we run the same number of classic epochs as the maximum number of epochs seen in any experiment with reduced epochs. This will distinguish whether or not improved results (if any) are due to applying reduced epochs, or just a higher number of epochs.

Table 6.2: Baseline results for accuracy. ET: Epochs Trained, BRE: Best Result Epoch

Model	Validation	Complete	Color	Combined	Crop	Resolution	ET	BRE
SqueezeNet	40.0	39.1	27.0	27.6	18.3	35.6	110	29
ResNet-152	61.6	60.4	42.3	35.3	24.0	53.6	50	11
EfficientNet-B3	73.4	74.8	33.3	28.0	31.0	45.3	170	143

For the obtained results we performed a significance test for comparing the methodologies with respect to the baseline using McNemar's test. We show in this subsection the significant results; for all the results we refer the reader to Annexed B. The nomenclature for significance is the following: if the p-value is greater than 0.05 it is not significant (ns); if the value is less than or equal to 0.05 it is represented as *; if it is less than or equal to 0.01 it is represented as **; if it is less than or equal to 0.001 it is represented as ***; and if it is less than or equal to 0.0001 it is represented as ****.

Table 6.3: Significant Test Results versus Baseline.

Network	**Methodology**	**Dimension**	**Validation**	**Complete**	**Color**	**Combined**	**Crop**	**Resolution**	**Significance**
EfficientNet	Start Point	Start Point	68.0	69.7	30.0	24.3	27.7	38.3	***
EfficientNet	Baseline	Baseline	73.4	74.8	33.3	28.0	31.0	45.3	
EfficientNet	Linear-PE	Crop	69.4	71.2	34.3	25.7	32.3	40.0	**
EfficientNet	Linear-PE	Quantization	70.2	66.6	30.7	25.0	30.0	41.7	***
EfficientNet	Linear-PE	Resolution	69.0	69.6	30.3	26.0	28.7	39.7	***
EfficientNet	Adaptive-PR	Combined	71.2	70.0	32.7	30.0	36.7	43.7	***
EfficientNet	Adaptive-PR	Resolution	69.4	69.4	29.7	26.0	30.3	38.7	***

The complete tables are available in Annexed B and we also provide detailed graphics which compares the training methodologies for each Neural Network in Annexed A.

6.1.6. Entropy Proportion

In Figures 6.15, 6.16, 6.17 we provide boxplots for the entropy quotients obtained from the best checkpoints of each methodology. In the columns we have the reduction dimension used for training and in the rows the reduction dimension used for obtaining the quality-reduced image according to the MEPIs Calculation described in Section 5.4. We highlight that a lower score indicates more robust (or laconic [9]) Image Classification, i.e., that the model successfully classified more images of lower quality.

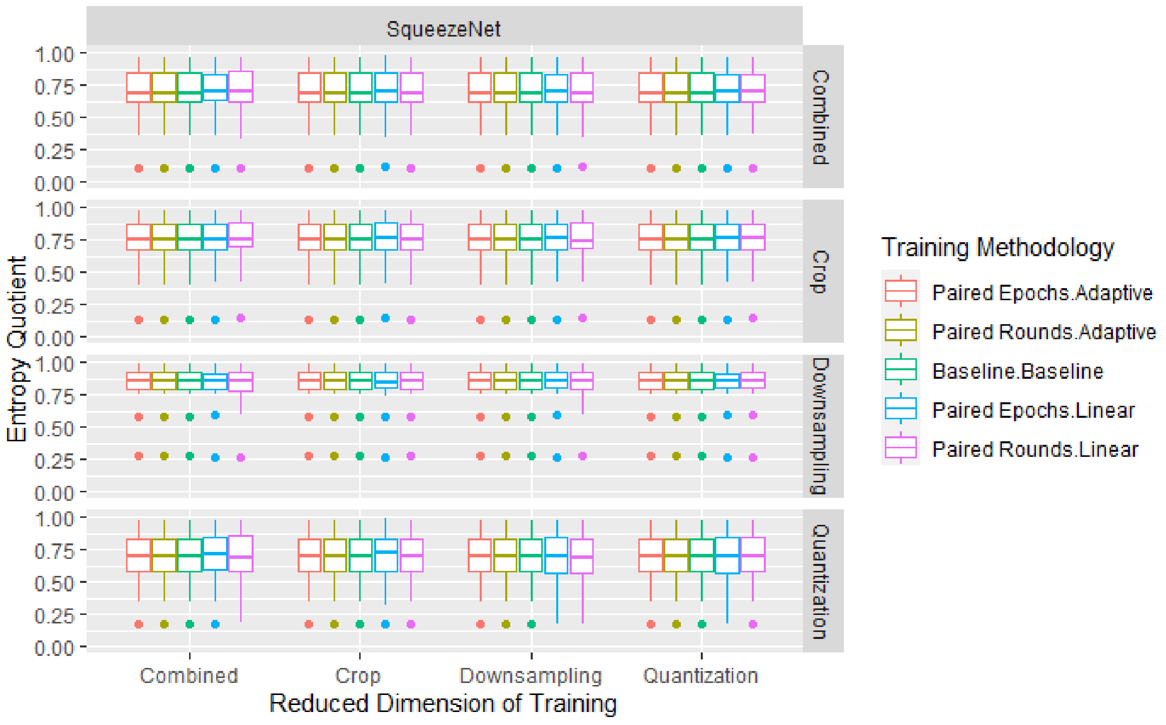


Figure 6.15: Entropy Quotient of SqueezeNet's Performance.

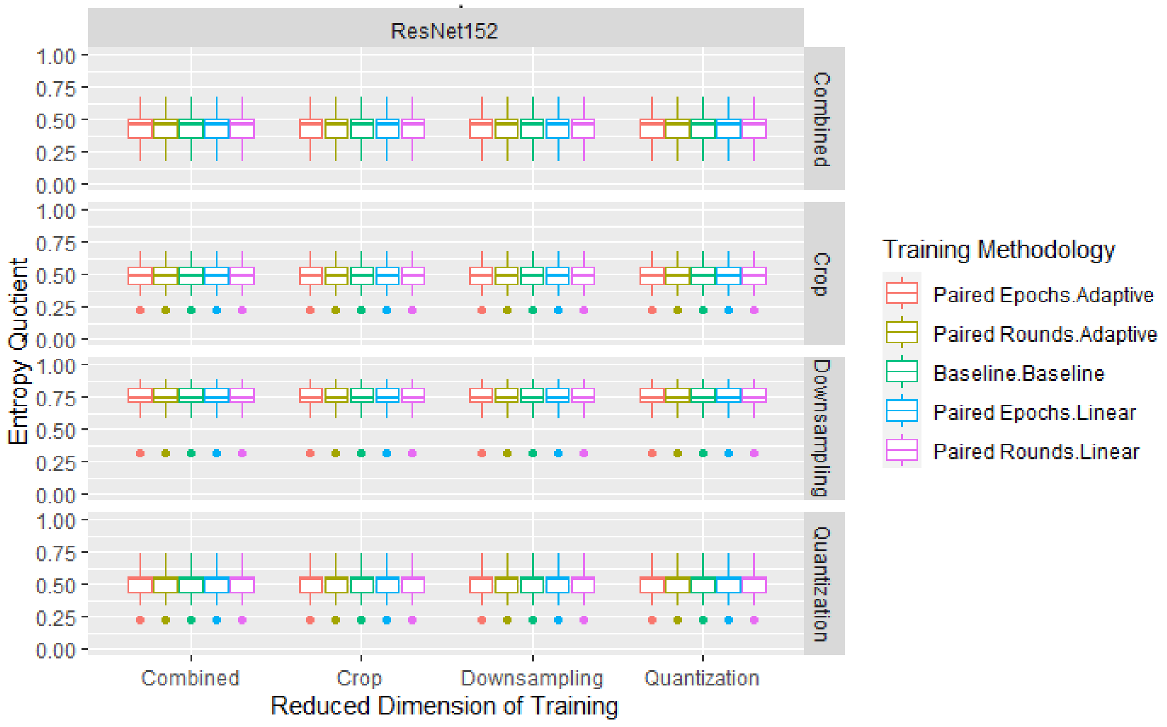


Figure 6.16: Entropy Quotient of ResNet's Performance.

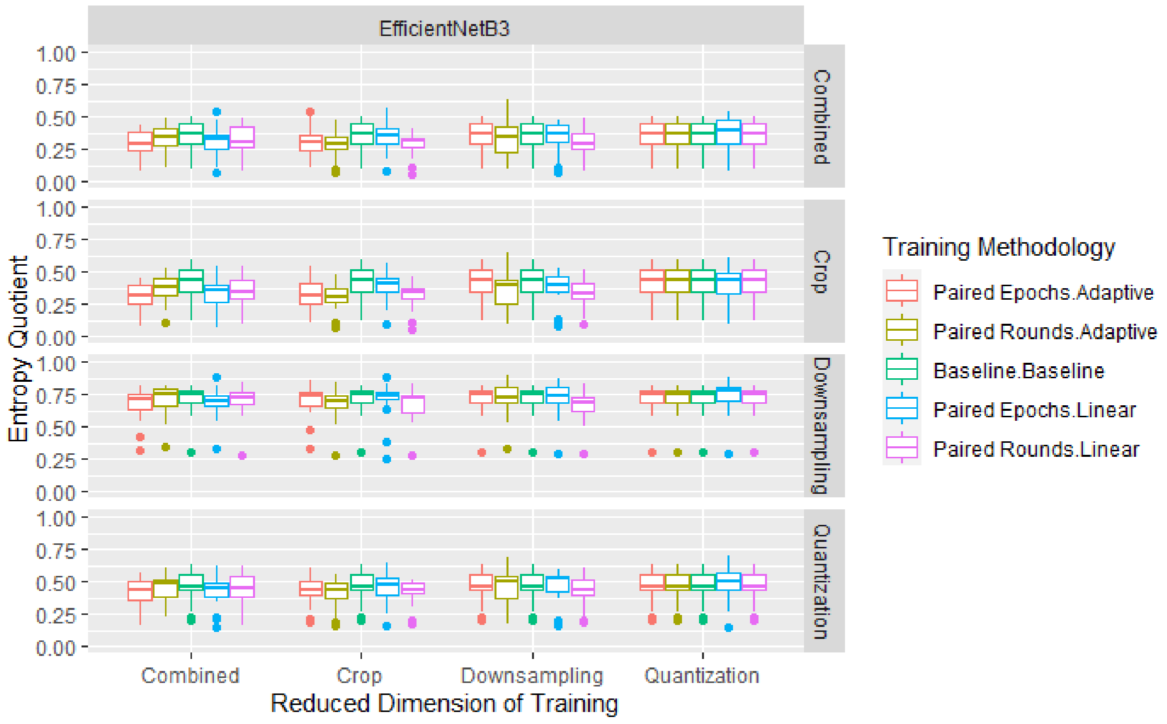


Figure 6.17: Entropy Quotient of EfficientNet's Performance.

From these results we see that some methodologies lower the entropy quotient with respect to the baseline, yielding a correct classification over lower quality images. This is particularly true for EfficientNet, though little or no difference is seen for ResNet nor with SqueezeNet.

We also provide in Table 6.4 the significant results, obtained comparing the results with the start point in Table 6.4 and with the baseline in Table 6.5 performing a Wilcoxon signed-rank test. We use the same significance levels as before. All significant results were for EfficientNet, and involved a reduction in the entropy ratios.

Table 6.4: EfficientNet Entropy Ratio Significance versus Start Point.

Methodology	**Training Reduction**	**Applied Reduction**	**Significance**
Adaptive-PE	Combined	Quantization	*
Linear-PE	Combined	Downsampling	*
Linear-PE	Resolution	Crop	*
Adaptive-PR	Combined	Combined	*
Adaptive-PR	Resolution	Combined	*
Linear-PR	Resolution	Quantization	*
Linear-PE	Combined	Combined	**
Linear-PE	Combined	Quantization	**
Adaptive-PR	Crop	Downsampling	**
Adaptive-PR	Crop	Quantization	**
Adaptive-PR	Resolution	Crop	**
Linear-PR	Combined	Crop	**
Linear-PR	Crop	Downsampling	**
Linear-PR	Crop	Quantization	**
Linear-PR	Resolution	Combined	**
Adaptive-PE	Combined	Combined	***
Adaptive-PE	Combined	Crop	***
Adaptive-PE	Combined	Downsampling	***
Adaptive-PE	Crop	Combined	***
Adaptive-PE	Crop	Crop	***
Linear-PE	Combined	Crop	***
Adaptive-PR	Combined	Crop	***
Adaptive-PR	Crop	Combined	***
Adaptive-PR	Crop	Crop	***
Linear-PR	Crop	Combined	***
Linear-PR	Crop	Crop	***
Linear-PR	Resolution	Crop	***
Linear-PR	Resolution	Downsampling	***

Table 6.5: EfficientNet Entropy Ratio Significance Comparison versus Baseline.

Methodology	**Training Reduction**	**Applied Reduction**	**Significance**
Adaptive-PE	Crop	Combined	*
Adaptive-PE	Combined	Combined	**
Adaptive-PE	Crop	Crop	**
Adaptive-PR	Crop	Combined	**
Adaptive-PE	Combined	Crop	***
Adaptive-PR	Crop	Crop	***

6.2. ImageNet

In this section we report the obtained results on the full ImageNet dataset. Given the costs of training models over the full ImageNet dataset, with experiments taking several weeks, we choose a single Neural Network and a training methodology, which surpassed the baseline. The selected methodology was based on the obtained results on HumaNet in terms of performance, significance and efficiency, in terms of execution time. Specifically, we chose the Paired Rounds, Linear variant, Combined dimension methodology for testing.

6.2.1. Training Methodology Tested

In Figure 6.18 we show the evolution of the accuracy on the validation set for training the start point model (no image reductions); the best result is highlighted with a vertical line. To boost the process of obtaining the start point we started the training with the checkpoint provided by the Pytorch library for the ImageNet dataset.

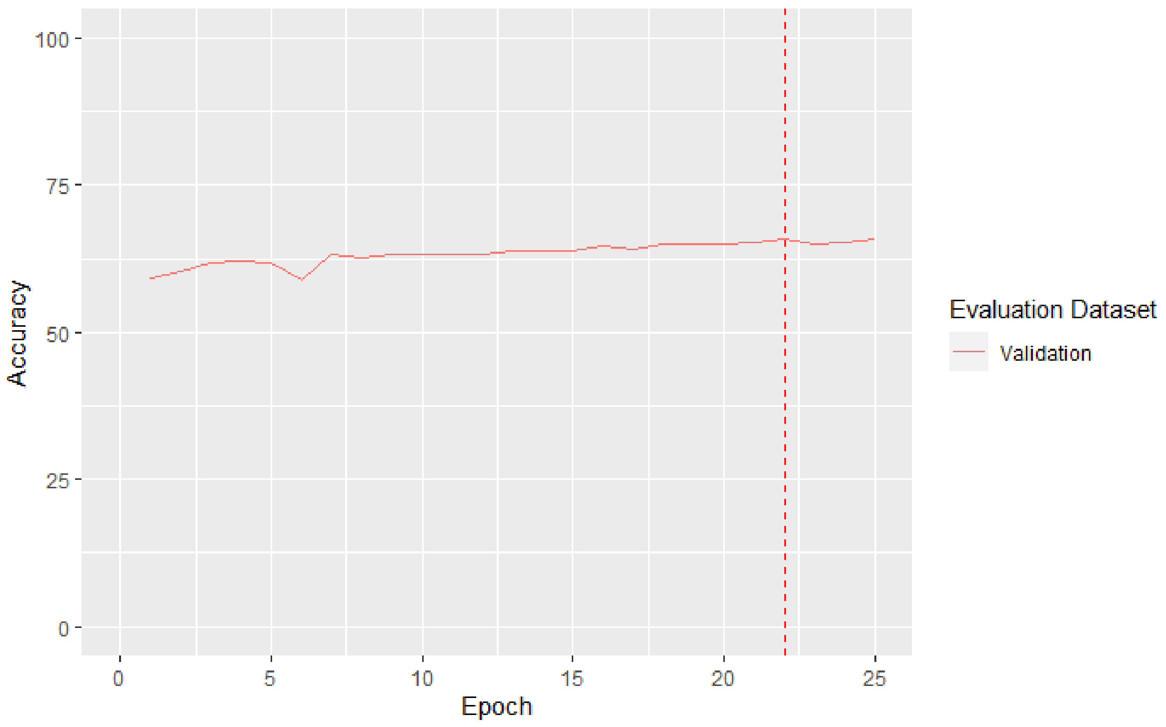


Figure 6.18: Start Point of EfficientNet-B3 on ImageNet.

In Figure 6.19 we show the evolution of the accuracy of the validation set and the best result is highlighted with a red vertical line. We also provide a gray vertical dotted line to indicate the starting point of the baseline and reduced methodology, and a gray vertical dashed line to indicate the epoch in which the maximum value was achieved in the baseline. We see that continuing from the start point, both the baseline and the reduced methodology continue to improve performance as more epochs are applied. The fact that the baseline improves suggests that a higher patience value might be more suitable for EfficientNet. We also see clear drops in performance for reduced epochs, which bounce back (often above the baseline) upon applying the next classical epoch. The best overall result is achieved with the reduced methodology.

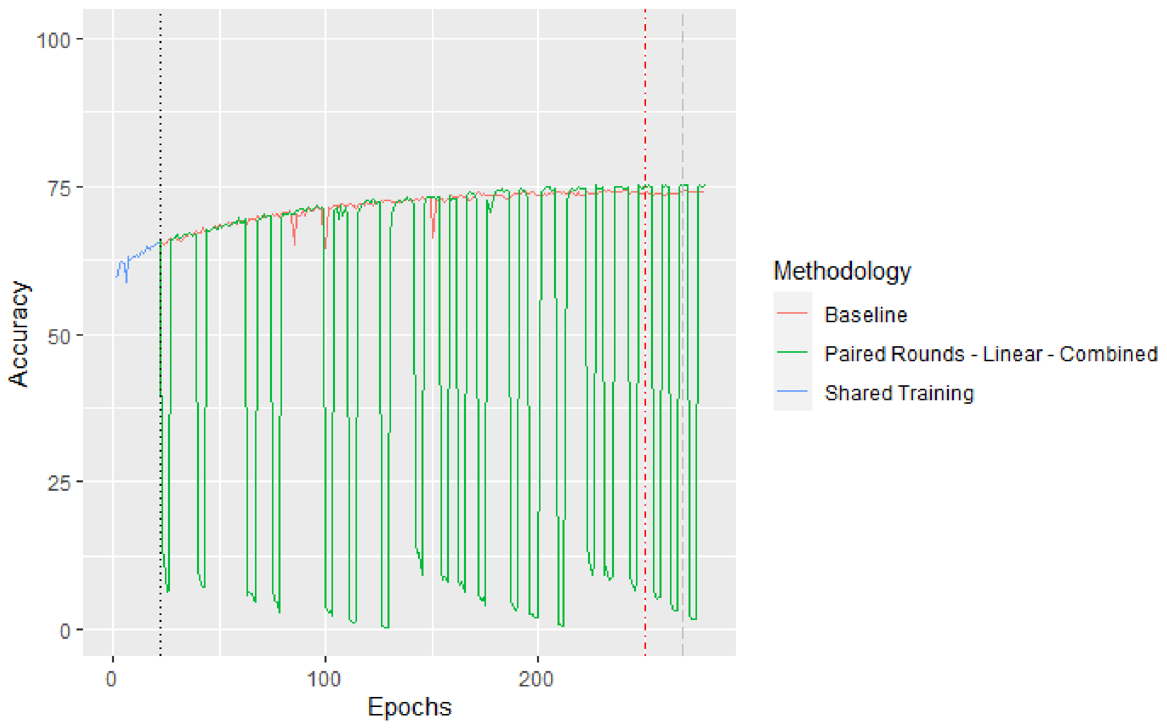


Figure 6.19: Performance evolution of EfficientNet during training ImageNet validation set, linear reduction, paired rounds and baseline.

In Table 6.6 we summarize the validation results and test accuracy results obtained for the methodology on the ImageNet dataset. In the table we can see from the obtained results that the pattern obtained in HumaNet, in terms of the drop and subsequent increase of the performance, persists and that the baseline validation result is surpassed by the proposed methodology.

Table 6.6: EfficientNet results on ImageNet with significance versus Baseline.

Methodology	**Validation**	**Test**	**Significance**
Start Point	65.804	63.314	****
Baseline	74.364	71.614	
Paired Rounds Linear	75.348	72.200	***

6.2.2. Entropy Proportion

In Figure 6.20 we report the entropy quotient obtained from the best checkpoint of the chosen methodology. As can be seen the chosen methodology has a lower ratio in the Combined, Crop and Quantization reductions, meaning that it can reduce the image size more in the process of obtaining the *MEPI*s. Each column of the figure represents the reduction dimension applied for obtaining the *MEPI*s. For the Central Tendency Measures of the boxplot we refer the reader to Annexed C.

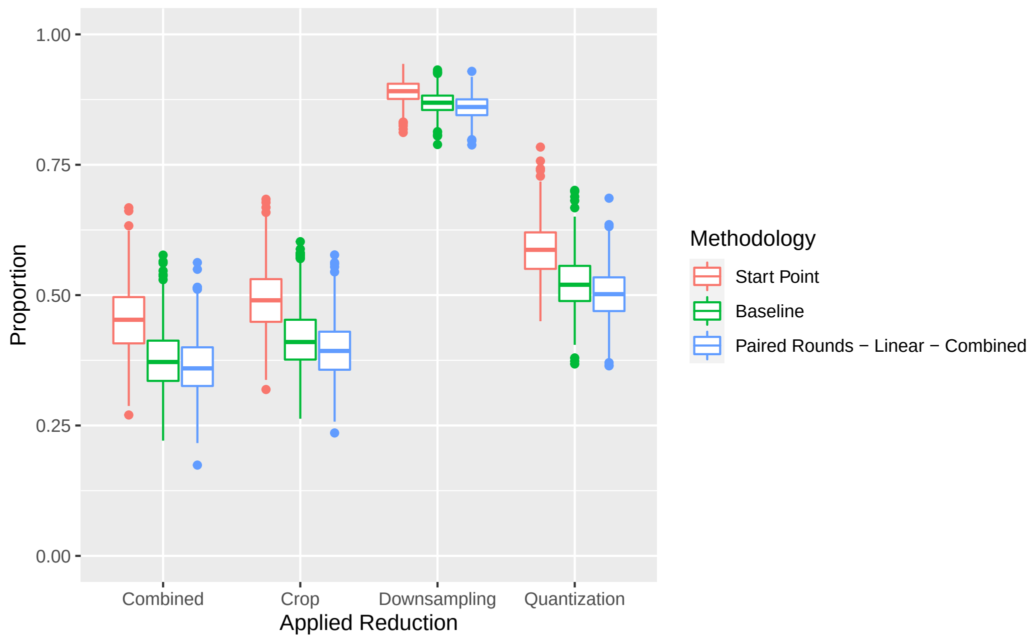


Figure 6.20: Entropy Quotient of EfficientNet's trained on full ImageNet.

Chapter 7

Conclusions

We conclude with a summary of our contributions, a review of our hypothesis in light of the experiments presented in the previous chapter, discussion of challenges and limitations, and finally, future directions.

7.1. Summary

In this work, we address the Image Classification task whereby, given an image, the classifier has to predict the class of object depicted by the image. In particular, we proposed a training methodology for Neural Networks classifiers with the goal of improving performance under challenging conditions (e.g. low resolution, sliced image, reduction of colors), studying the performance of the resulting classifier on both complete images (without reductions) and quality reduced images.

The quality reductions used were downsampling, quantization and crop, and the combination of these three quality reductions, which were extended from Carrasco et al. [9] to Pytorch. We trained these quality reductions with two methods for building a quality-reduced image, Linear and Adaptive; the former used a deterministic increase in the level of the quality reduction and the latter calculated the level of quality reduction based on feedback from the Neural Network. The sequence of classic and quality-reduced epochs was determined by the Paired Epochs and Paired Rounds variants of both methods, where the former applied a sequence of pairs of classic and quality-reduced epochs until there was no improvement of the performance over a fixed number of epochs, while the latter applied a sequence of reduced epochs at a given quality reduction level followed by a sequence of classic epochs, where each sequence ended when no improvement was seen for a fixed number of epochs.

We evaluated each methodology over three Convolutional Neural Network architectures and we compared the performance with our baseline in terms of the accuracy achieved in each dataset and the quality reduction of the predicted classes.

7.2. Review of Hypothesis

According to the results we obtained in our work, we have seen that the performance over a test set of complete images (without quality reductions) is similar to our methodologies in terms of accuracy, where no meaningful improvements are seen for SqueezeNet and ResNet. However, a slight and significant improvement is seen for EfficientNet, in terms of accuracy, in the HumaNet dataset, increasing from 74.8 to 74.9, when tested over full-quality images (see

Table B.3), being an improvement for the best result. With respect to a test set of images with quality reductions, our methodology outperformed the baseline in some datasets with quality reductions and this improvement was more notable in the case of EfficientNet: in particular in Adaptive-PR, for the Crop dimension results on datasets Color, Combined and Crop outperformed the baseline. Comparing the methodologies we propose, the improvements in performance were more notable with the Adaptive construction of the reduced images. The experiments performed validate our hypothesis for the case of EfficientNet architecture, but not for SqueezeNet nor ResNet.

We tested our Neural Network classifiers trained with our proposed methodology calculating the *Minimal Entropy Positive Image* as described in Section 5.4, from which we obtained, in general, a median entropy ratio lower than the baseline's median. In the methodology tested on ImageNet we also see this trend (see Figure 6.20), in particular for the MEPIs obtained using Crop, there is a similar standard deviation 0.056 and a difference of 0.017 between the median of the entropy ratio between the Baseline and our Methodology. Therefore, depending on the applied methodology, the reduced entropy quotient may be lower in some cases.

From the experiments made, we consider that applying quality reductions to training images may help the Neural Network to train with the most important regions of an image and thus improve performance, depending on the applied quality reduction.

7.3. Review of Objectives

We choose ImageNet as our image dataset for our study, along with three Convolutional Neural Networks: SqueezeNet, ResNet and EfficientNet. We established a baseline considering accuracy and entropy ratios as comparative metrics.

For training Convolutional Neural Networks with quality reductions we designed two algorithms for applying quality reductions during training: Linear and Adaptive. The application of these reduced quality images were done in two types of sequences interleaving quality reduced and full quality batches: Paired Rounds and Paired Epochs.

We improved our results with respect to the baseline in one of the three architectures. In particular, for the ImageNet dataset and our training methodology over the EfficientNet architecture, we saw an improvement from 71.6 to 72.2 in terms of accuracy, in the ImageNet dataset. Thus we think our general objective and specific objectives were accomplished.

7.4. Challenges and Limitations

A challenge we had to face was the fact that without experiments we could not know how our training methodology would perform, so a purely theoretical approach would not give us clarity about our training methodology. In this scenario, the different Neural Network architectures, variants of training methodologies proposed, the dataset sizes and the time needed to train classifiers of this kind, led us to the challenge of considering the available hardware for prioritizing our experiments. We partially addressed this challenge by developing a smaller image dataset, which allowed us to run a broader range of preliminary experiments. However, even over the smaller dataset, we constantly needed to prioritize the next experiments to run, which was itself a challenge, as it was difficult to predict beforehand which approaches were most promising.

We identified a difference between the achieved performance in the different Neural Networks; thus a limitation of our methodology is that different architectures and hyperparameters (such as learning rate, weight decay) may affect performance. During training, the learning rate value evolves, so reduced epochs may have a learning rate with a lower value if the algorithm decides to make more classic epochs beforehand. Ideally we could run experiments varying the learning rate, but per the previous discussion, searching over hyperparameters would have added a prohibitive cost to our experiments.

We also consider that only taking into account the validation accuracy over complete images as a guide for the current performance of the network during training may limit the progression of the training versus considering all sets of images, including reduced images.

7.5. Future Work

For future work, we consider tests on other datasets and also to prepare other reduced datasets to test our methodology. We also think it would be important to weigh each dataset's performance to use as a guide for the training progress in order to take into account both reduced and complete images. For reduced images, we consider that a set of reduced images may be important to use as a compressed version of the images for training with the essential features under certain quality reductions. We also consider that it would be of interest to test our methodology with different hyperparameters (such as learning rate, weight decay or optimizer), which may have different impact in the process of training depending on the Neural Network architecture.

During our work we used one Neural Network for improving the performance on both full-quality and reduced images, having different performance in each methodology for the same dataset, leaving the question of how to choose the best methodology. One proposal would be to train a Neural Network which discriminates between a full-quality and a reduced image and its quality reduction, and to classify it with a more specialized Neural Network trained with that quality reduction.

Bibliography

[1] Krizhevsky, A., Sutskever, I., and Hinton, G. E., "Imagenet classification with deep convolutional neural networks," Advances in neural information processing systems, pp. 1097–1105, 2012, https://dl.acm.org/doi/10.5555/2999134.2999257.

[2] Simonyan, K. and Zisserman, A., "Very deep convolutional networks for large-scale image recognition.," 2014, https://arxiv.org/abs/1409.1556.

[3] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A., "Going deeper with convolutions.," 2014, https://arxiv.org/abs/1409.1556.

[4] He, K., Zhang, X., Ren, S., and Sun, J., "Deep residual learning for image recognition.," Proceedings of the IEEE conference on computer vision and pattern recognition., pp. 770–778, 2016, https://arxiv.org/abs/1512.03385.

[5] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K., "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size.," 2016, https://arxiv.org/abs/1602.07360.

[6] Tan, M. and Le, Q., "Efficientnet: Rethinking model scaling for convolutional neural networks," International Conference on Machine Learning, pp. 6105–6114, 2019, https://arxiv.org/abs/1905.11946.

[7] Geirhos, R., Temme, C. R. M., Rauber, J., Schütt, H. H., Bethge, M., and Wichmann, F. A., "Generalisation in humans and deep neural networks.," 2018, https://arxiv.org/abs/1808.08750.

[8] Dodge, S. and Karam, L., "A Study and Comparison of Human and Deep Learning Recognition Performance Under Visual Distortions.," 2017, https://arxiv.org/abs/1705.02498.

[9] Carrasco, J., Hogan, A., and Pérez, J., "Laconic Image Classification: Human vs. Machine Performance," Proceedings of the 29th ACM International Conference on Information & Knowledge Management, 2020, doi:https://dx.doi.org/10.1145/3340531.3411984.

[10] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P., "SMOTE: Synthetic Minority Over-sampling Technique.," Journal of artificial intelligence research.., vol. 16, pp. 321–357, 2002, doi:https://dx.doi.org/10.1613/jair.953.

[11] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P., "Deepfool: a simple and accurate method to fool deep neural networks.," Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2574–2582, 2016, https://arxiv.org/abs/1511.04599.

[12] Shorten, C. and Khoshgoftaar, T. M., "A survey on image data augmentation for deep learning.," Journal of Big Data, vol. 6, no. 1, pp. 1–48, 2019, doi:https://dx.doi.org/1

0.1186/s40537-019-0197-0.

[13] Antoniou, A., Storkey, A., and Edwards, H., "Data augmentation generative adversarial networks.," 2017, https://arxiv.org/abs/1711.04340.

[14] Ahmed, N., Natarajan, T., and Rao, K. R., "Discrete cosine transform.," IEEE transactions on Computers., vol. 100, no. 1, pp. 90–93, 1974, doi:https://dx.doi.org/10.1109/T-C.1974.223784.

[15] Kullback, S. and Leibler, R. A., "On information and sufficiency.," The annals of mathematical statistics., vol. 22, no. 1, pp. 79–86, 1951, http://www.jstor.org/stable/2236703.

[16] Tieleman, T. and Hinton., G., "Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude.," 2012, https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[17] Kingma, D. P. and Ba, J., "Adam: A method for stochastic optimization.," 2014, https://arxiv.org/abs/1412.6980.

[18] Powell, M. J. D., "An efficient method for finding the minimum of a function of several variables without calculating derivatives.," The Computer Journal., vol. 7, no. 2, pp. 155–162, 1964, doi:https://dx.doi.org/10.1093/comjnl/7.2.155.

[19] Bengio, Y., "Practical recommendations for gradient-based training of deep architectures.," Neural networks: Tricks of the trade., pp. 437–478, 2012, https://arxiv.org/abs/1206.5533.

[20] Wei, J. and Zou, K., "Eda: Easy data augmentation techniques for boosting performance on text classification tasks.," 2019, https://arxiv.org/abs/1901.11196.

[21] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huan, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L., "Imagenet large scale visual recognition challenge.," International journal of computer vision., vol. 115, no. 3, pp. 211–252, 2016, https://arxiv.org/abs/1409.0575.

[22] O'Hara, S. and Draper, B. A., "Introduction to the bag of features paradigm for image classification and retrieval.," 2011, https://arxiv.org/abs/1101.3354.

[23] Frid-Adar, M., Diamant, I., Klang, E., Amitai, M., Goldberger, J., and Greenspan, H., "GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification.," Neurocomputing., vol. 321, pp. 321–331, 2018, https://arxiv.org/abs/1803.01229.

[24] He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R., "Masked autoencoders are scalable vision learners.," 2021, https://arxiv.org/abs/2111.06377.

[25] Hu, J., Shen, L., Albanie, S., Sun, G., and Wu, E., "Squeeze-and-Excitation Networks," Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 7132–7141, 2018, https://arxiv.org/abs/1709.01507.

[26] Hendrycks, D. and Dietterich, T., "Benchmarking neural network robustness to common corruptions and perturbations.," 2019, https://arxiv.org/abs/1903.12261.

[27] Taori, R., Dave, A., Shankar, V., Carlini, N., Recht, B., and Schmidt, L., "Measuring robustness to natural distribution shifts in image classification.," 2020, https://arxiv.org/abs/2007.00644.

ANNEXES

Annexed A. Methodologies Comparison

In this section we discuss each methodology applied over each dataset. We also provide graphics in which we distinguish each training methodology's performance over a dataset in which each row indicates the reduction applied during training for the methodologies shown. We also distinguish the shared training we used as a starting point for each training methodology.

A.1. SqueezeNet

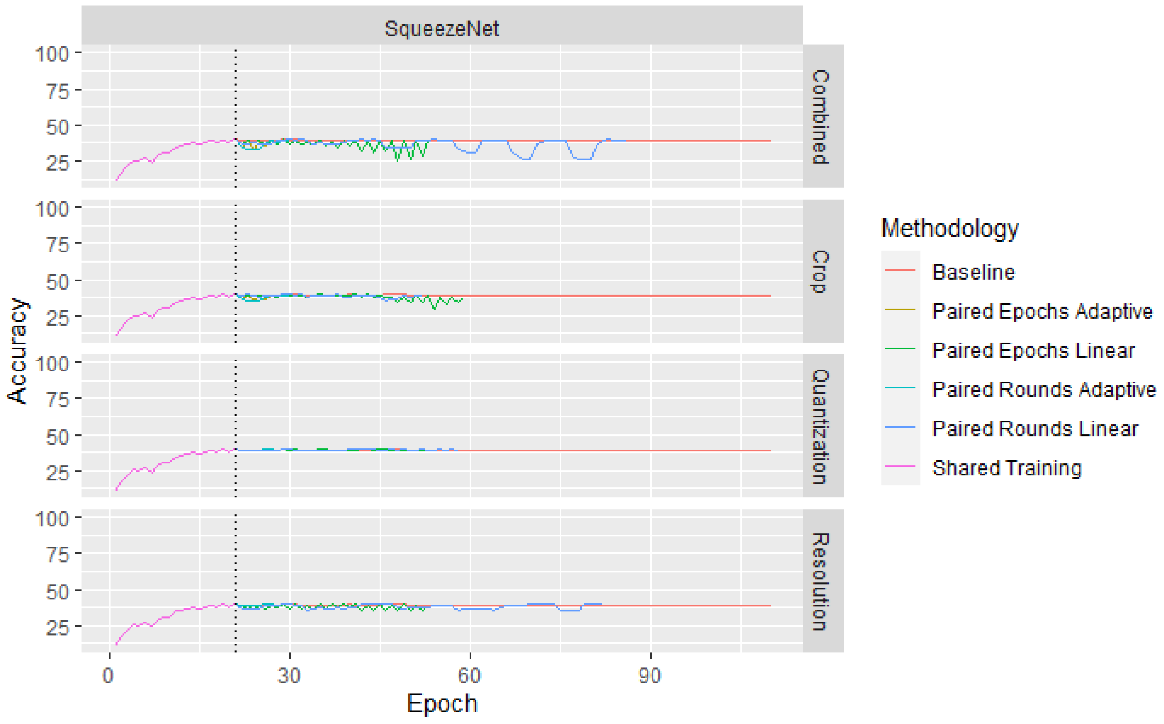


Figure A.1: Performance progression of each methodology on the Validation dataset of SqueezeNet.

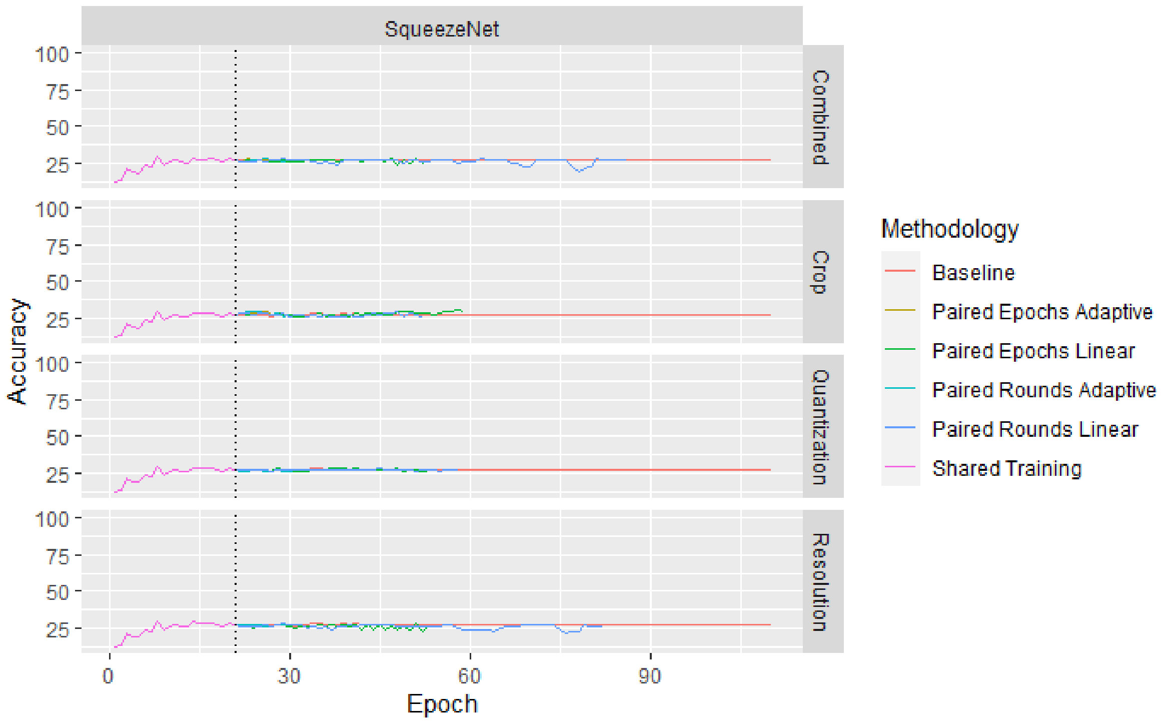


Figure A.2: Performance progression of each methodology on the Combined dataset of SqueezeNet.

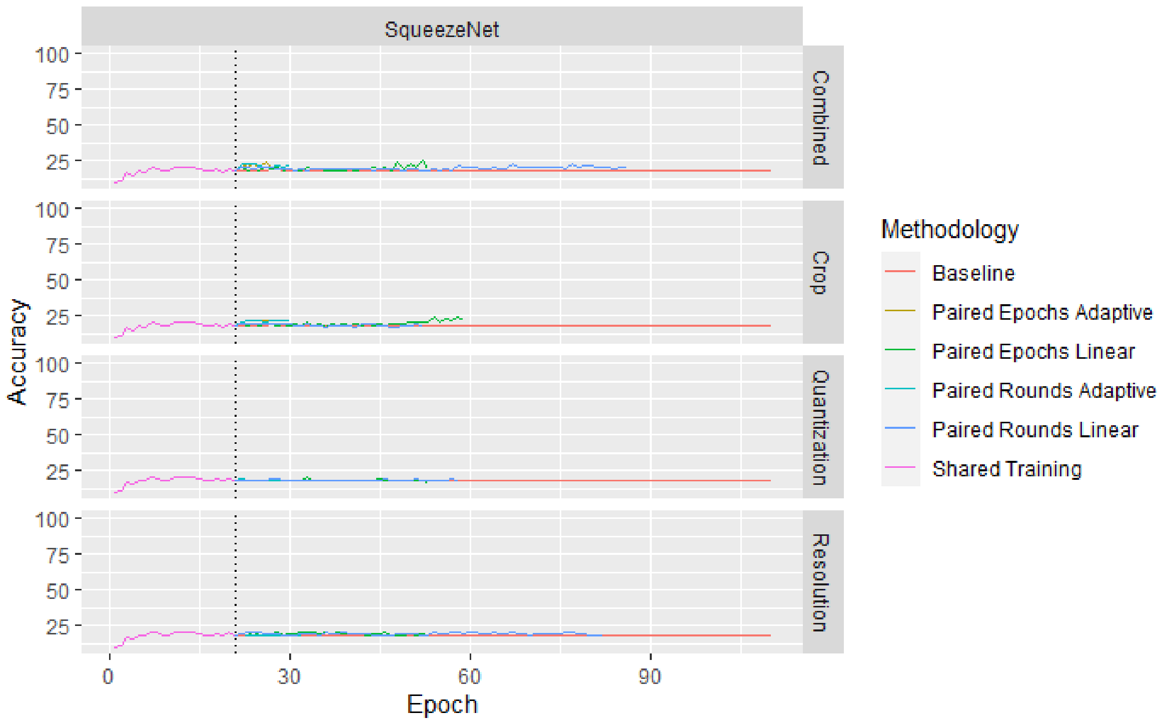


Figure A.3: Performance progression of each methodology on the Crop dataset of SqueezeNet.

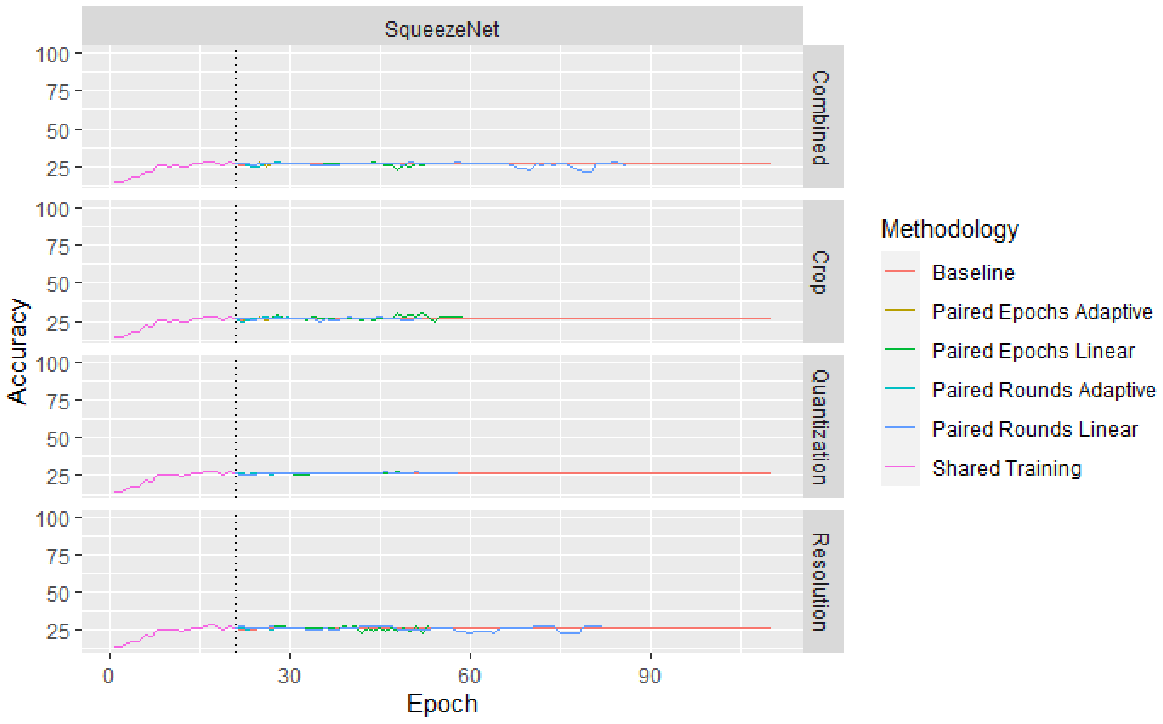


Figure A.4: Performance progression of each methodology on the Color dataset of SqueezeNet.

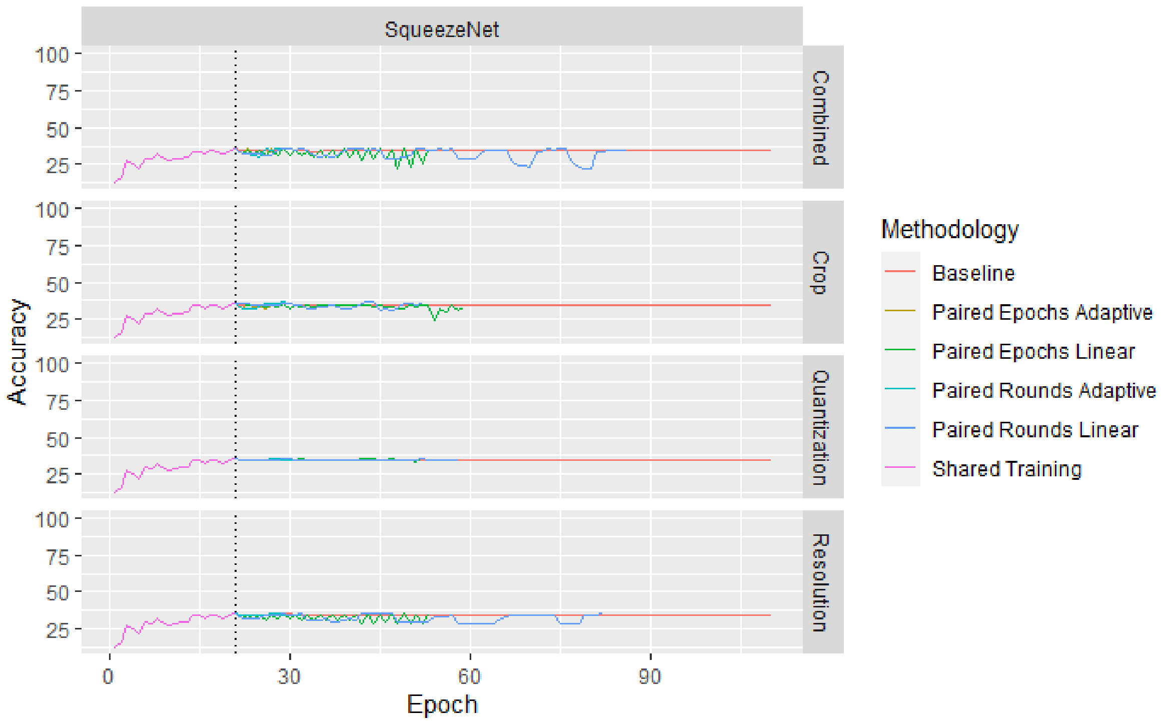


Figure A.5: Performance progression of each methodology on the Resolution dataset of SqueezeNet.

In this Neural Network the obtained results perform similar to the baseline, but Paired Epochs Linear has better performance than the baseline in the Crop dataset if it is being trained with Crop and Combined reductions, as can be seen in Figure A.3.

A.2. ResNet

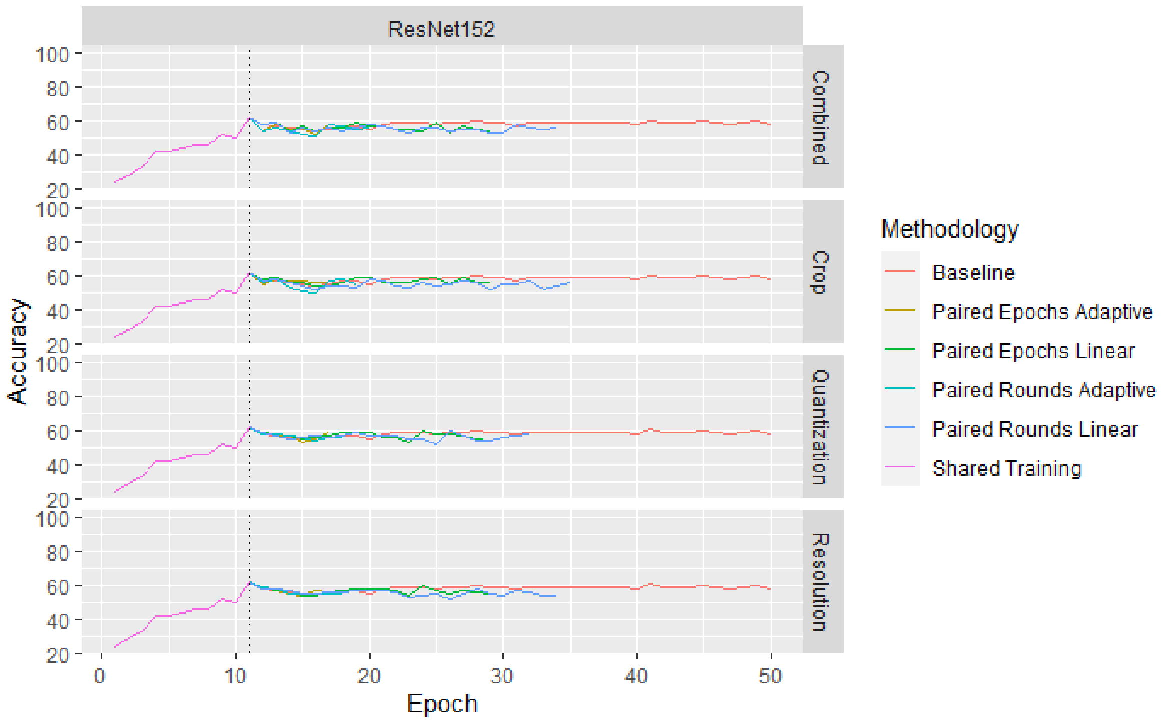


Figure A.6: Performance progression of each methodology on the Validation dataset of ResNet.

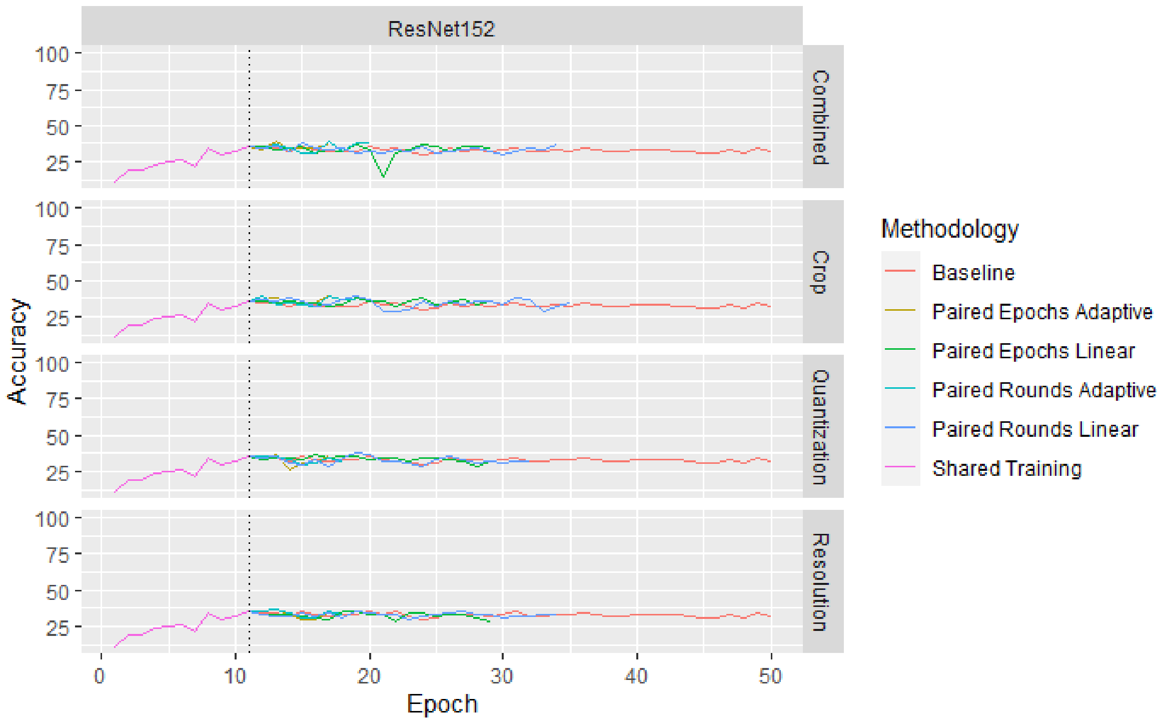


Figure A.7: Performance progression of each methodology on the Combined dataset of ResNet.

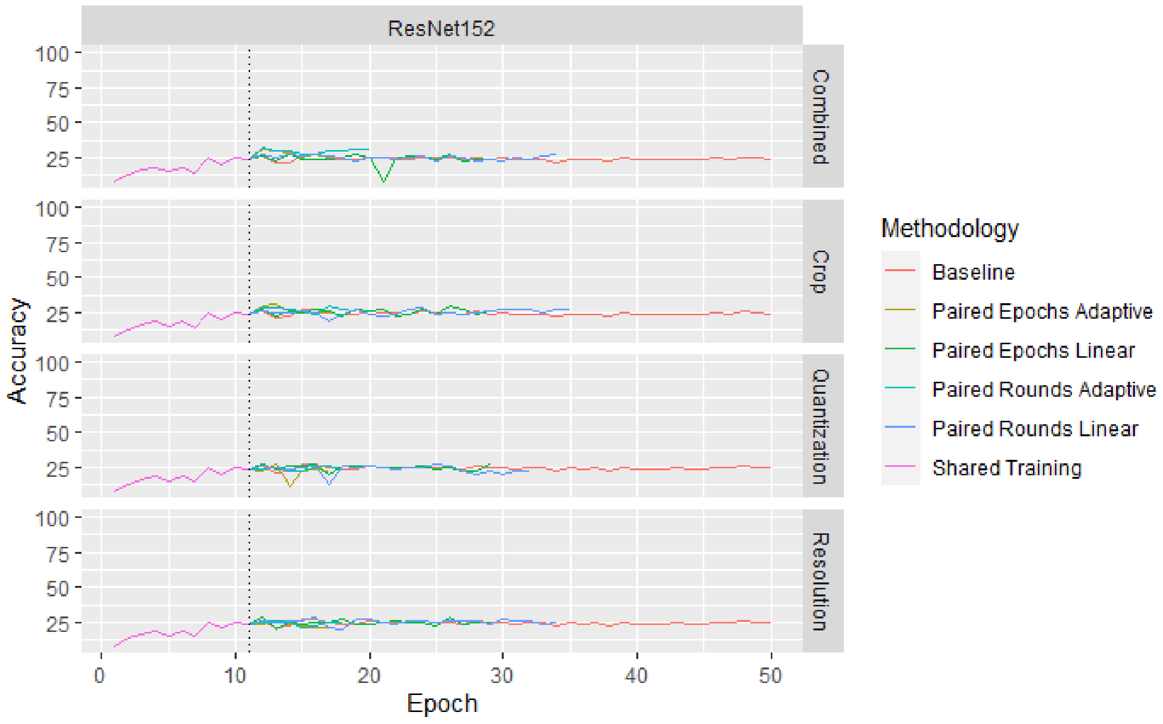


Figure A.8: Performance progression of each methodology on the Crop dataset of ResNet.

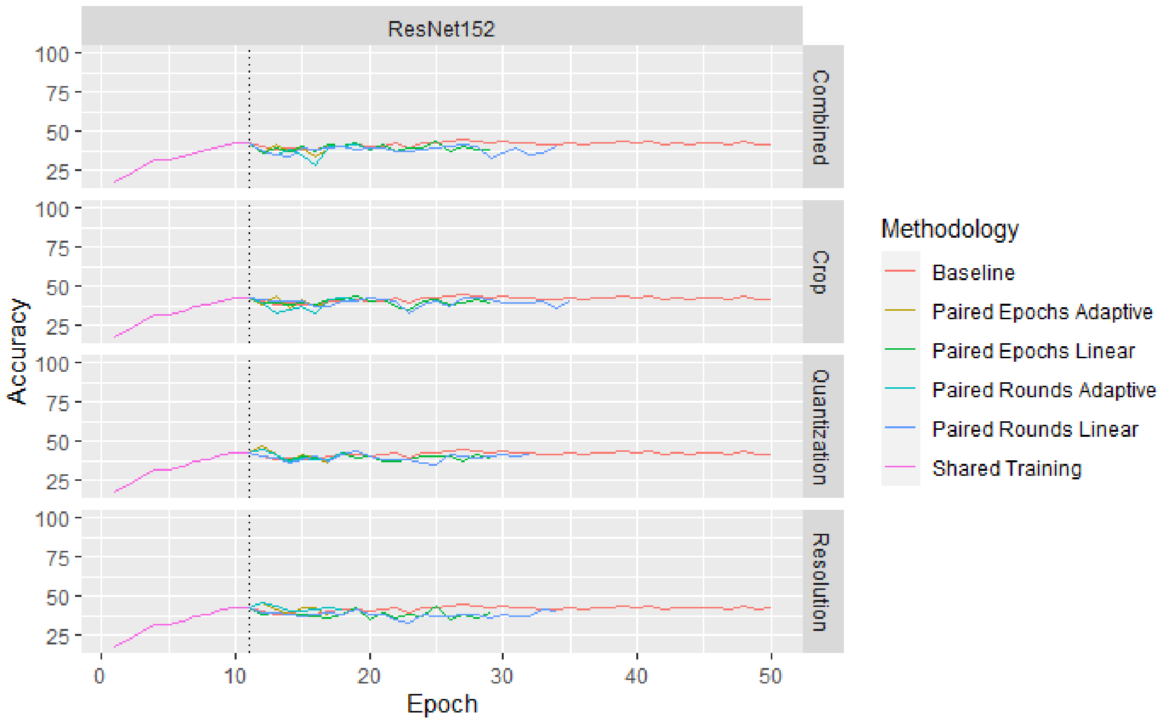


Figure A.9: Performance progression of each methodology on the Color dataset of ResNet.

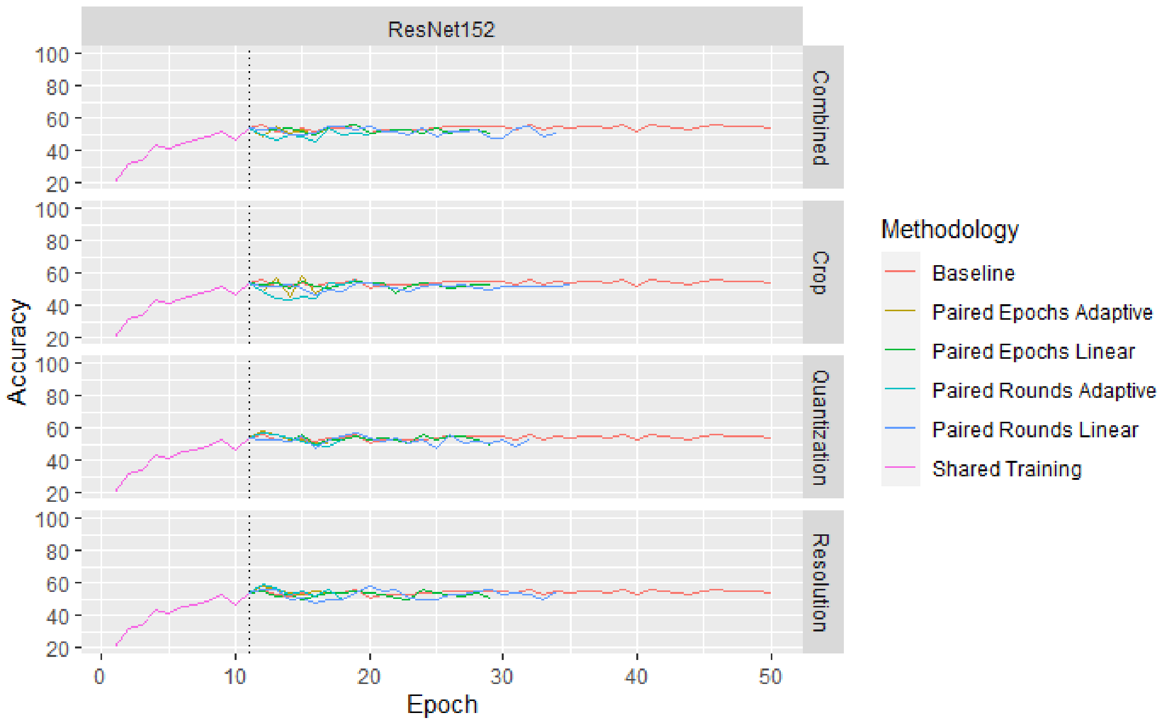


Figure A.10: Performance progression of each methodology on the Resolution dataset of ResNet.

In this Neural Network, no methodology outperformed the accuracy of the baseline, and thus it did not save any checkpoint, so training ended sooner. Nevertheless, the Paired Epochs Linear methodology outperformed the baseline in the Combined (Figure A.7) and Crop (Figure A.8) datasets, when the Neural Network is trained on Combined and Crop reduced images.

A.3. EfficientNet

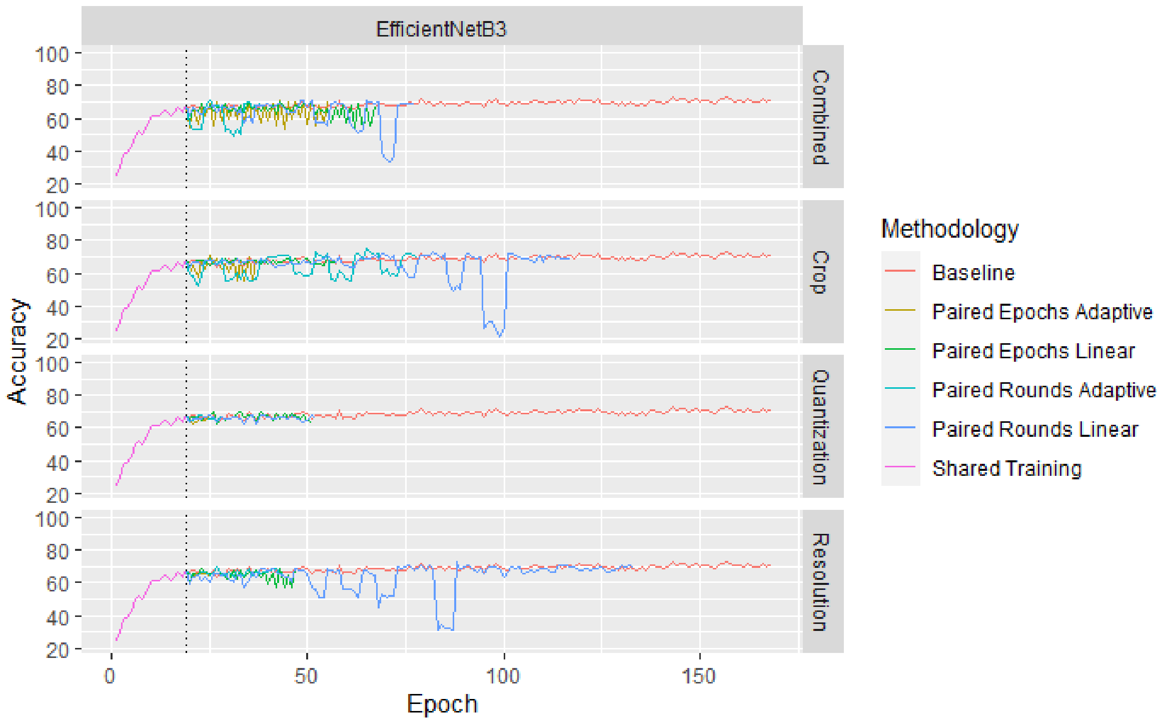


Figure A.11: Performance progression of each methodology on the Validation dataset of EfficientNet.

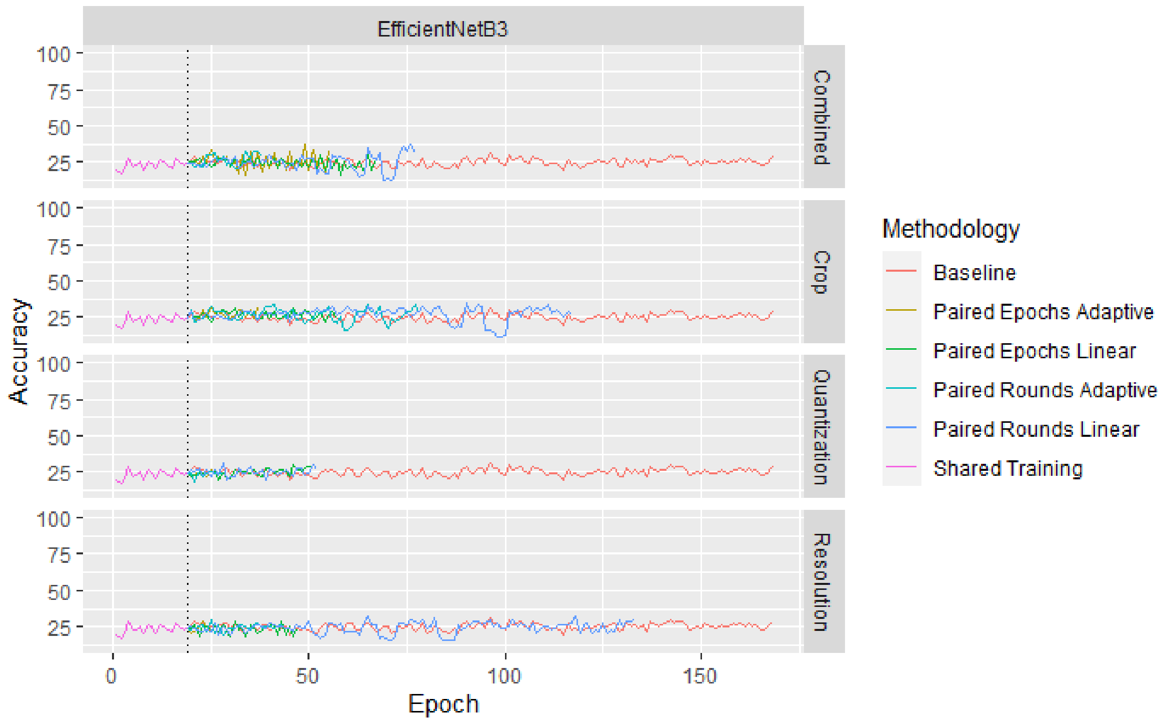


Figure A.12: Performance progression of each methodology on the Combined dataset of EfficientNet.

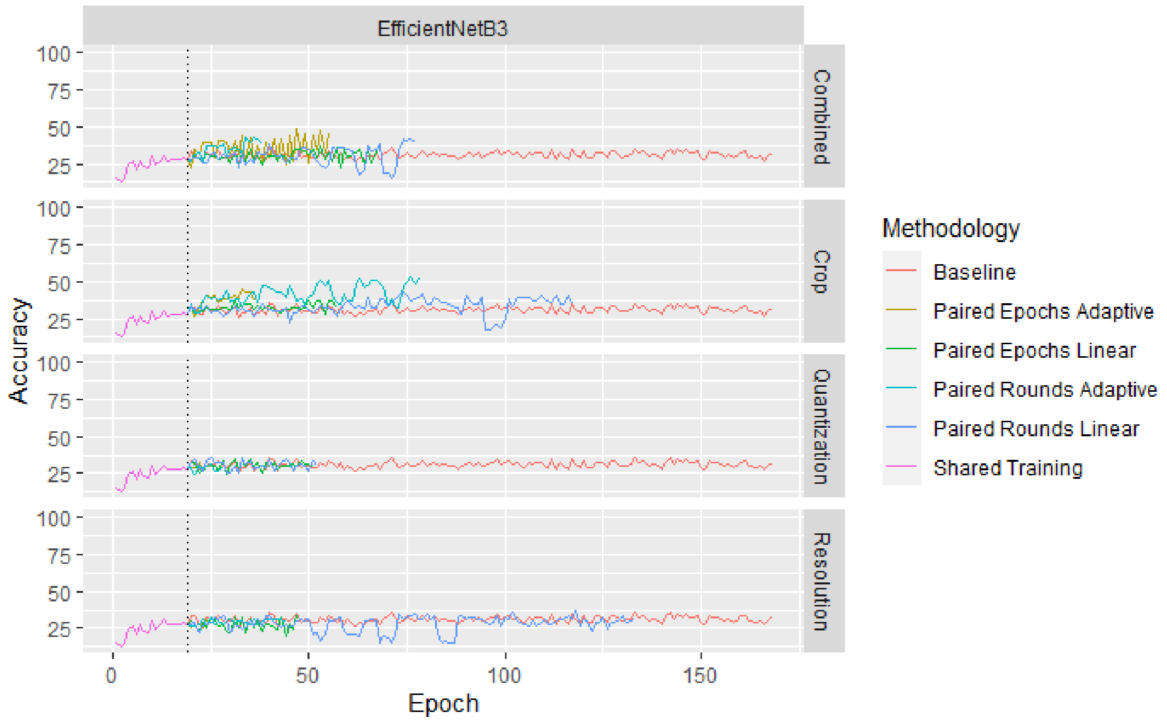


Figure A.13: Performance progression of each methodology on the Crop dataset of EfficientNet.

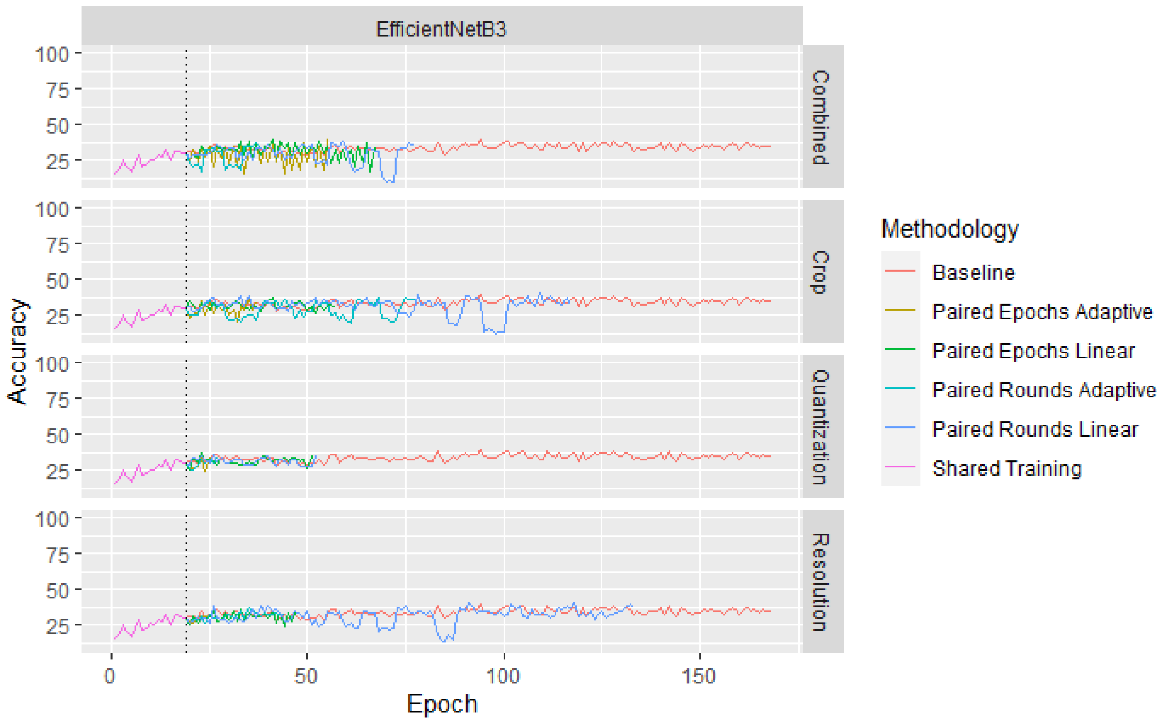


Figure A.14: Performance progression of each methodology on the Color dataset of EfficientNet.

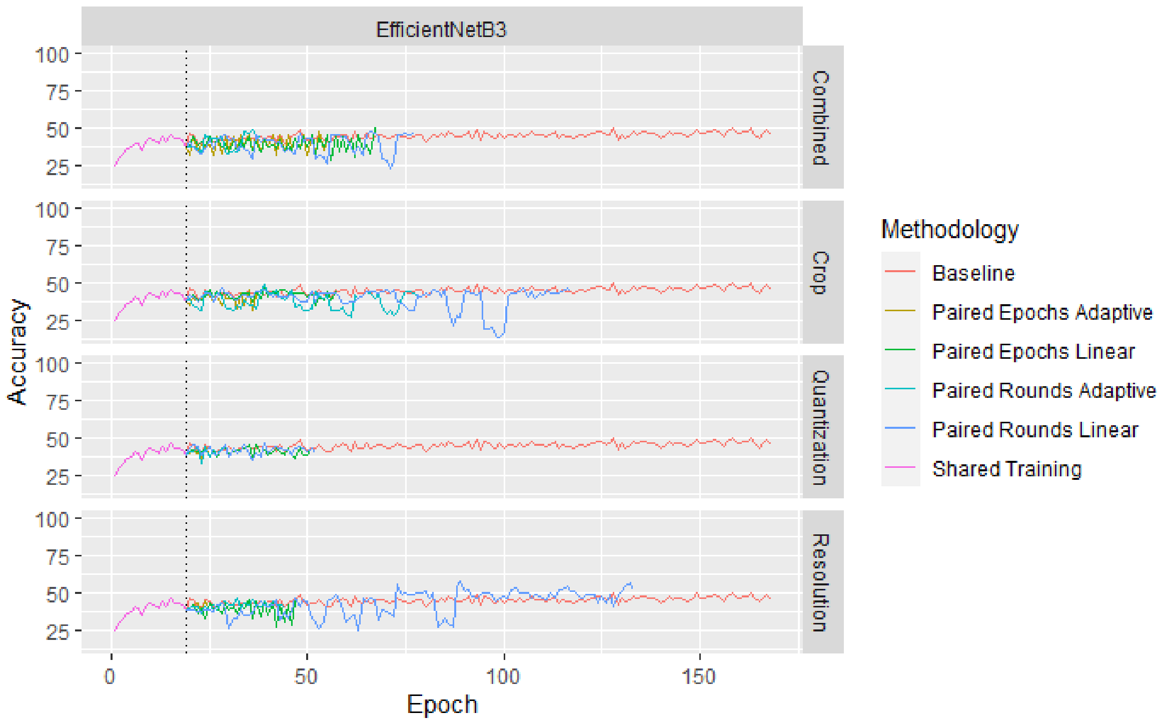


Figure A.15: Performance progression of each methodology on the Resolution dataset of EfficientNet.

In this Neural Network we can observe that the performance on the validation dataset is similar to the baseline (Figure A.11), but in the reduced datasets (Figures A.12, A.13, A.15) we can see that our methodology outperforms the baseline performance.

We can also observe that an adaptive methodology, which performs more guided reductions, may improve the performance for the crop dataset (Figure A.13), when the Neural Network is trained with Combined and Crop reductions.

Annexed B. Performance and Significance

Here we present the results obtained on each methodology on HumaNet considering its significance with respect to the baseline. Please note that in Table B.2 for ResNet, the model given by the reduced methodologies was precisely the same as the starting point model, and the baseline model, and thus no significance is reported.

Table B.1: SqueezeNet results on HumaNet.

Methodology	Dimension	Validation	Complete	Color	Combined	Crop	Resolution	Significance
Adaptive-PE	Combined	39.8	39.0	26.7	27.3	18.3	35.3	ns
Adaptive-PE	Crop	39.8	39.0	26.7	27.3	18.3	35.3	ns
Adaptive-PE	Quantization	39.8	39.0	26.7	27.3	18.3	35.3	ns
Adaptive-PE	Resolution	39.8	39.0	26.7	27.3	18.3	35.3	ns
Linear-PE	Combined	40.2	39.0	27.3	28.0	18.0	36.0	ns
Linear-PE	Crop	40.4	38.7	26.3	28.0	19.7	35.0	ns
Linear-PE	Quantization	40.2	39.2	27.0	27.7	18.3	35.7	ns
Linear-PE	Resolution	40.2	39.2	27.3	27.7	18.3	36.0	ns
Adaptive-PR	Combined	39.8	39.0	26.7	27.3	18.3	35.3	ns
Adaptive-PR	Crop	39.8	39.0	26.7	27.3	18.3	35.3	ns
Adaptive-PR	Quantization	39.8	39.0	26.7	27.3	18.3	35.3	ns
Adaptive-PR	Resolution	39.8	39.0	26.7	27.3	18.3	35.3	ns
Linear-PR	Combined	40.0	39.2	26.3	27.3	19.7	34.7	ns
Linear-PR	Crop	39.8	39.0	26.7	27.3	18.3	35.3	ns
Linear-PR	Quantization	40.0	39.1	27.0	27.7	18.3	36.0	ns
Linear-PR	Resolution	40.2	39.5	28.0	27.0	18.0	35.7	ns
Start Point	Start Point	39.8	39.0	26.7	27.3	18.3	35.3	ns

Table B.2: ResNet results on HumaNet.

Methodology	Dimension	Validation	Complete	Color	Combined	Crop	Resolution	Significance
Adaptive-PE	Combined	61.6	60.4	42.3	35.3	24.0	53.7	
Adaptive-PE	Crop	61.6	60.4	42.3	35.3	24.0	53.7	
Adaptive-PE	Quantization	61.6	60.4	42.3	35.3	24.0	53.7	
Adaptive-PE	Resolution	61.6	60.4	42.3	35.3	24.0	53.7	
Linear-PE	Combined	61.6	60.4	42.3	35.3	24.0	53.7	
Linear-PE	Crop	61.6	60.4	42.3	35.3	24.0	53.7	
Linear-PE	Quantization	61.6	60.4	42.3	35.3	24.0	53.7	
Linear-PE	Resolution	61.6	60.4	42.3	35.3	24.0	53.7	
Adaptive-PR	Combined	61.6	60.4	42.3	35.3	24.0	53.7	
Adaptive-PR	Crop	61.6	60.4	42.3	35.3	24.0	53.7	
Adaptive-PR	Quantization	61.6	60.4	42.3	35.3	24.0	53.7	
Adaptive-PR	Resolution	61.6	60.4	42.3	35.3	24.0	53.7	
Linear-PR	Combined	61.6	60.4	42.3	35.3	24.0	53.7	
Linear-PR	Crop	61.6	60.4	42.3	35.3	24.0	53.7	
Linear-PR	Quantization	61.6	60.4	42.3	35.3	24.0	53.7	
Linear-PR	Resolution	61.6	60.4	42.3	35.3	24.0	53.7	
Start Point	Start Point	61.6	60.4	42.3	35.3	24.0	53.7	

Table B.3: EfficientNet results on HumaNet.

Methodology	**Dimension**	**Validation**	**Complete**	**Color**	**Combined**	**Crop**	**Resolution**	**Significance**
Adaptive-PE	Combined	71.4	73.9	35.3	28.3	42.0	46.3	ns
Adaptive-PE	Crop	71.2	72.5	34.0	31.3	39.0	42.0	ns
Linear-PE	Combined	69.8	73.4	38.0	27.7	31.3	42.0	ns
Adaptive-PR	Crop	75.0	74.9	35.7	34.0	47.7	45.0	ns
Linear-PR	Combined	71.4	72.4	34.7	30.7	36.0	46.7	ns
Linear-PR	Crop	72.6	74.4	31.7	31.7	39.3	45.3	ns
Linear-PR	Resolution	72.6	73.8	31.3	26.3	32.3	54.0	ns
Linear-PE	Crop	69.4	71.2	34.3	25.7	32.3	40.0	**
Linear-PE	Quantization	70.2	66.6	30.7	25.0	30.0	41.7	***
Linear-PE	Resolution	69.0	69.6	30.3	26.0	28.7	39.7	***
Adaptive-PR	Combined	71.2	70.0	32.7	30.0	36.7	43.7	***
Adaptive-PR	Resolution	69.4	69.4	29.7	26.0	30.3	38.7	***
Linear-PR	Quantization	68.0	69.7	30.0	24.3	27.7	38.3	***
Adaptive-PE	Quantization	68.0	69.7	30.0	24.3	27.7	38.3	***
Adaptive-PE	Resolution	68.0	69.7	30.0	24.3	27.7	38.3	***
Adaptive-PR	Quantization	68.0	69.7	30.0	24.3	27.7	38.3	***
Start Point	Start Point	68.0	69.7	30.0	24.3	27.7	38.3	***

Annexed C. ImageNet Entropy-Ratio Central Tendency Measures

In this Annexed we present the Central Tendency Measures obtained in the ImageNet Entropy-Ratio.

Table C.1: Central Tendency Measures of ImageNet Entropy-Ratio.

Dimension	**Methodology**	**Median**	**Mean**	**Standard Deviation**
Combined	Start Point	0.453	0.453	0.061
Combined	Baseline	0.372	0.375	0.058
Combined	Proposed	0.360	0.364	0.057
Crop	Start Point	0.490	0.491	0.058
Crop	Baseline	0.410	0.414	0.056
Crop	Proposed	0.393	0.394	0.056
Downsampling	Start Point	0.891	0.890	0.021
Downsampling	Baseline	0.869	0.869	0.022
Downsampling	Proposed	0.861	0.860	0.022
Quantization	Start Point	0.587	0.586	0.051
Quantization	Baseline	0.520	0.523	0.049
Quantization	Proposed	0.502	0.503	0.049