UNIVERSIDADE BEIRA INTERIOR
Engenharia

A zero-shot learning method for recognizing objects using low-power devices

Cristiano Pires Patrício

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor João Carlos Raposo Neves
Co-orientador: Prof. Doutor Hugo Pedro Martins Carriço Proença

Covilhã, junho de 2021

Acknowledgements

Firstly, I would like to express my deepest gratitute to my supervisor João Neves, for his invaluable guidance, support, and useful advises during these months. I am also grateful to him for the valuable knowledge transfer in each meeting, that inherently fostered the thriving of the work carried out in this dissertation, whilst improving my skills.

A word of gratitute is also addressed to NOVA-LINCS Laboratory for providing the hardware resources where the work of this dissertation was developed.

To my family, especially my parents João and Helena I would like to express my sincere gratitute for their continuous encouragement and support throughout my studies. My special thanks to my brother Gonçalo for all the amazing moments and the great laughs.

To my girlfriend Carolina, words are not enough to express how grateful I am for all your support, inspiration and endless love during this journey. My work always goes further thanks to the extra motivation that you place in me.

Finally, I would like to thank my classmates, in particular Ricardo, Pedro and Diandre for all the good times we spent in these two years, but mostly for the great friendship. I also wish to thank my laboratory partner Luís for all the good moments and conversations in the hundred of coffee breaks.

Resumo

O *Zero-Shot Learning* (ZSL) tem sido uma área de interesse crescente devido ao seu paradigma revolucionário que visa simular o comportamento humano na tarefa de reconhecimento de objetos que nunca foram vistos anteriormente. Os modelos de ZSL devem ser capazes de reconhecer classes de objetos que nunca tenham sido vistos durante o treino do classificador, tendo apenas como auxílio para a previsão de classes desconhecidas, descrições textuais das mesmas.
Apesar da vasta literatura existente em torno da temática do ZSL, são poucos os trabalhos que avaliam o desempenho computacional dos métodos desenvolvidos, no que diz respeito ao tempo dispendido na fase de inferência. Até à data, nenhum trabalho avaliou o impacto do uso de arquiteturas menos complexas e com menor custo computacional nos métodos de ZSL, para além da arquitetura padrão de facto *ResNet101*. Além do mais, a viabilidade de implementar os métodos de ZSL em aplicações do mundo real, particularmente fazendo uso de dispositivos de baixa capacidade computacional, ainda não foi estudada.
Assim, esta dissertação faz a avaliação de diferentes métodos de ZSL no que respeita ao impacto do uso de arquiteturas menos complexas de redes neuronais convolucionais no desempenho geral dos métodos de ZSL. Desta forma, é possível ficar ciente do comportamento dos métodos de ZSL em cenários reais, principalmente quando implementados em dispositivos de baixa capacidade computacional.
Os resultados obtidos demonstraram que o impacto no valor da precisão dos métodos de ZSL não é significativo quando são adotadas arquiteturas menos complexas para efeitos de extração de caraterísticas das imagens, sendo possível inferir que os métodos de ZSL são capazes de operar em tempo real em dispositivos de baixa capacidade computacional.

Palavras-chave

Zero-shot learning, Dispositivos de baixa capacidade computacional, Modelos generativos

Resumo alargado

A área da aprendizagem automática divide-se em três paradigmas fundamentais: aprendizagem supervisionada, aprendizagem não-supervisionada e aprendizagem por reforço. Relativamente à aprendizagem supervisionada, no que respeita ao problema de classificação de imagens, a abordagem típica é treinar um classificador com uma grande quantidade de images do conjunto de treino, onde existem exemplos para todas as classes que se pretendem prever, posteriormente. No entanto, há situações onde pode não ser possível recolher uma vasta amostra de exemplos de treino para todas as classes. Um exemplo clássico é a classificação de espécies raras de pássaros, por exemplo, onde recolher uma grande amostra de imagens para todas as classes pode ser díficil. Todavia, pode acontecer também que determinado classificador de imagens seja confrontado com uma imagem de teste pertencente a uma classe que nunca apareceu durante o treino. Neste caso, a previsão da classe será sempre incorreta, uma vez que o classificador não tem capacidade de generalizar para além das classes que apareceram no treino. É neste sentido que surge um novo paradigma, com o objetivo de colmatar os problemas discutidos anteriormente. O Zero-Shot Learning (ZSL) é um paradigma de aprendizagem promissor no qual não são necessários exemplos de imagens para as classes que se pretendem prever, ou seja, as classes de treino e as classes de teste são disjuntas. Esta abordagem é possível graças à introdução de um espaço adicional, denominado espaço semântico ou espaço de atributos. Cada classe será descrita através de uma descrição semântica, que pode ser em formato de texto ou numérico (atributos). É através destes atributos, ou anotações textuais, que os algoritmos de ZSL aprendem a generalizar para as classes que nunca foram vistas pelo classificador, uma vez que a informação semântica das classes de teste pode ser acedida durante o treino. Dada uma imagem de teste, uma abordagem típica é fazer uso de um classificador para prever os atributos da imagem, que posteriormente são comparados com os atributos existentes no espaço dos atributos (ou semântico), por forma a fazer corresponder a classe do atributo que for mais similar ao atributo previsto.

Durante a revisão da literatura, foram estudados os conceitos fundamentais acerca da temática do ZSL, nomeadamente, a caraterização dos diferentes espaços semânticos, os principais métodos de ZSL, os desafios e problemas, os *datasets* mais usados e os protocolos de avaliação. As abordagens usadas para resolver o problema do ZSL incluem métodos baseados em classificadores de atributos, métodos baseados em projeção e, mais recentemente, métodos generativos.

Apesar da vasta literatura existente em torno da temática do ZSL, são escassos os trabalhos que avaliam as abordagens propostas em termos de eficiência computacional, mais concretamente, uma análise do desempenho dos modelos em termos de tempo de inferência em cada uma das fases, nomeadamente a fase de extração de caraterísticas e a fase de previsão da classe. Além do mais, a grande maioria dos trabalhos existentes usa *Convolutional Neural Networks* (CNN) para a extração de caraterísticas visuais da imagem, onde a arquitetura *ResNet101* se tornou o padrão de facto. No entanto, e até à data, ainda ninguém avaliou o impacto que arquiteturas menos complexas e com menor custo com-

putacional, como a *MobileNet* ou a *MobileNetV2*, teriam no desempenho geral das abordagens ZSL, assim como ainda ninguém estudou como explorar estas arquitecturas para produzir sistemas de reconhecimento de objetos capazes de operar em tempo real em dispositivos de baixa capacidade computacional, como por exemplo o Raspberry Pi e o Jetson Nano. É neste sentido que, nesta dissertação, fornecemos um extenso estudo comparativo do desempenho computacional de várias abordagens ZSL fazendo uso de diferentes tipos de arquiteturas, a operar em dispositivos de baixa capacidade computacional. Os resultados obtidos permitiram concluir que o uso de uma arquitetura mais leve para fins de extração de caraterísticas não representa um impacto significativo no desempenho dos modelos de ZSL, indicando que o uso de dispositivos como o Jetson Nano e o Raspberry Pi é viável em ambientes de produção.

Além disso, propomos também uma nova abordagem de ZSL, que resultou da fusão de duas estratégias promissoras. Os resultados evidenciam que o método proposto é eficaz quando aplicado em *datasets* que incluem anotações de atributos visualmente interpretáveis, como é o caso do *dataset* CelebA. Por outro lado, quando aplicado aos *datasets* tradicionais do ZSL, não se registam melhorias.

Por fim, o trabalho resultante desta dissertação permitiu a escrita de um artigo científico onde é analisado o impacto do uso de arquiteturas mais leves no desempenho geral de vários métodos ZSL, bem como estudada a viabilidade de implementação dos métodos em dispositivos de baixa capacidade computacional.

Abstract

Zero-Shot Learning (ZSL) has been a subject of increasing interest due to its revolutionary paradigm that simulates human behavior in recognizing objects that have never seen before. The ZSL models must be capable of recognizing classes that do not appear during training, using only the provided textual descriptions of the unseen classes as an aid.
Despite the vast benchmarking around the ZSL paradigm, few works have assessed the computational performance of the developed strategy regarding inference time. Furthermore, no work has evaluated the effects of using different CNN architectures, such as lightweight architectures, apart from the *de facto* standard *ResNet101* architecture, and the feasibility of deploying zero-shot learning approaches in a real-world scenario, particularly when using low-power devices.
Consequently, in this dissertation, we carried out an extensive benchmarking toward analyzing the impact of using lightweight CNN architectures on ZSL performance, allowing us to perceive how the ZSL methods perform in real-world scenarios, mainly when run in low-power devices. Our experimental results demonstrate that the impact on the ZSL accuracy is not significant when a lightweight architecture is adopted, indicating the effectiveness of such low-power devices in performing ZSL methods.

Keywords

Zero-shot learning, Low-power devices, Generative models

Contents

List of Figures

List of Tables

Acronyms List

AI	Artificial Intelligence
ALE	Attribute Label Embedding
BoW	Bag-of-Words
CMT	Cross-Modal Transfer
CNN	Convolutional Neural Networks
ConSE	Convex Combination of Semantic Embeddings
CVAE	Conditional Variational Autoencoder
DAP	Direct Attribute Prediction
DeViSE	Deep Visual-Semantic Embedding Model
DNN	Deep Neural Network
ESZSL	Embarrassingly Simple Approach to Zero-Shot Learning
GAN	Generative Adversarial Network
GZSL	Generalized Zero-Shot Learning
IAP	Indirect Attribute Prediction
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
LatEm	Latent Embeddings for Zero-Shot Learning
ML	Machine Learning
PS	Proposed Split
ReLU	Rectified Linear Unit
SAE	Semantic AutoEncoder
SGAM	Semantic-Guided Attention Model
SGMA	Semantic-Guided Multi-Attention Model
SOC	Semantic Output Codes
SS	Standard Split
SSVM	Structured Support Vector Machine
SVM	Support Vector Machine
UBI	Universidade da Beira Interior
ZSL	Zero-Shot Learning

Chapter 1

Introduction

This document describes the work developed in the scope of the dissertation project for the attainment of the master's degree in Computer Science and Engineering at the University of Beira Interior (UBI). The first section presents the motivation and scope of the work, followed by section 1.2 where the problem statement and objectives are discussed. Finally, section 1.3 describes the organization of the remainder of this dissertation.

1.1 Motivation and Scope

Nowadays, Artificial Intelligence (AI) is part of our daily lives and has a wide range of application scenarios. Creating solutions that learn from experience to make accurate decisions or predictions is the focus of Machine Learning (ML), a branch of AI. One of the most popular applications of ML is image recognition - a technique used to identify and classify objects, people, and actions in images (data). The image recognition systems are based on one of the ML paradigms - supervised learning - and have specific applications in fingerprint identification systems, medical image analysis, and self-driving cars [14].

In supervised learning paradigm, the typical approach is to train a model with a large number of images (training data) containing all the classes intended to be recognized during the test phase. However, in many practical scenarios, it may not be possible to collect a wide variety of training data for each target class. For example, considering the problem of recognizing bird species, it may not be easy to obtain sufficient number of examples for each category. There could also be scenarios in which novel classes that are not covered by training examples appear in the testing phase. In addition, the learned model can only classify instances belonging to classes that are covered by the training instances, which results in the lack of ability to generalize to novel classes. To overcome these problems, methods under more promising learning paradigms have been proposed.
In machine learning, Zero-Shot Learning (ZSL), also known as zero-data learning, is a promising learning method, in which the classes covered by the training instances and the classes to be classified are disjoint [15]. Thereby, samples from unknown classes are classified based on a high-level description of these classes, such as textual description or a set of attributes that distinguishes those classes [15]. This assumption extends the applications of recognition systems significantly. The training phase does not depend on the entire set of classes to be recognized, contributing to reduce the human effort in collecting a large number of examples for each target class.

This dissertation addresses the problem of ZSL. In particular, it studies the feasibility of

using state-of-the-art ZSL approaches for recognizing objects not seen during the training phase, through a textual description, and using low-computational devices.

The research carried out in this dissertation is part of the project "Perception for a Service Robot" and the research grant UIDB/04516/2020/TRA/BIL/08 from the research unit NOVA Laboratory for Computer Science and Informatics (NOVA LINCS).

1.2 Problem Statement and Objectives

The main problem addressed in this work is the task of recognizing objects from their textual descriptions, based on a ZSL approach. ZSL intends to simulate human behavior in object recognition tasks even without ever having seen the object before, however providing only some type of auxiliary information, e.g., a textual description of the object. For example, a child will have no problem recognizing a zebra if she has seen horses before and has also read that the zebra looks like a horse but has black stripes. A concrete application where ZSL can be used is the development of a recognition module to be integrated in a moving robot that should be capable of classify unknown objects by using its textual description.

Thus, the main objective of this work is the development of a ZSL-based approach method capable of recognizing unknown objects using only its textual description. In addition, the developed strategy should be suitable to be embedded into a moving robot using low-power devices.

1.3 Dissertation Outline

The remainder of this dissertation is organized as follows: chapter 2 presents a detailed overview of the zero-shot learning taxonomy. Moreover, we describe and analyse a collection of state-of-the-art zero-shot methods and also evaluate their performance on the most widely used datasets in the task of zero-shot learning. Chapter 3 provides a description of the adopted methodology for the development of this work. This way, we provide the details of the developed evaluation framework for assessing ZSL methods in terms of computational performance, and at the end we describe our proposed Semantic-Guided Attention Model that aims to learn more discriminative visual features conditioned on the most dominant semantic attributes regarding the input image. In chapter 4 are presented an extensive benchmark regarding the computational performance of the ZSL methods and the results obtained after applying our proposed method on two datasets. Finally, conclusions are drawn in chapter 5, where the main contributions and the future work are also discussed.

Chapter 2

State of the Art

In this chapter, we provide an overview of relevant state-of-the-art methods to address zero-shot learning. We start by describing and enumerating different kinds of semantic spaces used in existing zero-shot learning works. In Section 2.2, we introduce an overview of zero-shot methods, where its general pipeline is described, followed by Section 2.3 that discusses the main challenges in these methods and some techniques to solve them. Section 2.4 summarizes the most widely used datasets in the task of zero-shot recognition. In section 2.5, we describe the evaluation metrics used to benchmark the zero-shot methods. Finally, we report the results of experiments in commonly used datasets to compare the performance of the methods.

2.1 Semantic Spaces

A semantic space can be defined as a multidimensional space in which each point represents the meaning of a word. The relationship between words is given by measuring the distance between them in the semantic space, which means that if two points are close to each other, they are semantically related.
In the context of zero-shot learning, semantic spaces contain semantic information about classes, denoted as class prototypes (vectors) with a specific dimension. It is common to refer to this semantic information as auxiliary information. The name "semantic" is employed here because the semantic space is the space of meaning, composed by a set of related nodes (that can be words or something with a numerical representation - vector) where the distance between two nodes represents its semantic relation [16]. Figure 2.1 depicts the semantic space constructed from the 85 dimensional classes prototypes of *Animals with Attributes 2* [1] dataset. The mapping from 85 to 2 dimensions was done using the t-SNE technique [17].
According to how a semantic space is constructed, we can distinguish them as (1) engineered semantic spaces and (2) learned semantic spaces [15]. Figure 2.2 shows the different kind of semantic spaces in a more depth-detailed categorization.

2.1.1 Engineered Semantic Spaces

The "engineered" term is used here to suggest that each dimension is designed by humans. There are different kinds of engineered semantic spaces, each of which is characterized by the uniqueness in the data source and how to construct them.

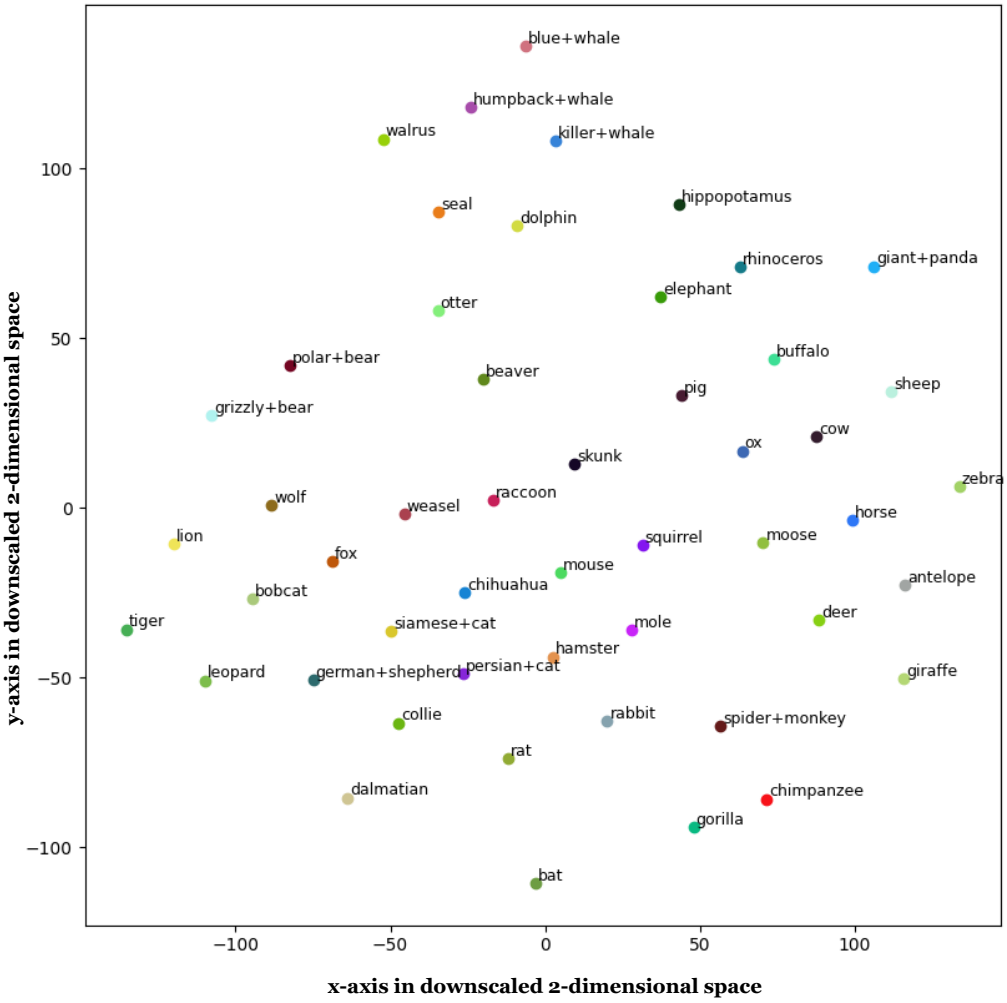


Figure 2.1: **T-SNE visualization of the semantic word space**. Each point corresponds to each of the 50 classes of the *Animals with Attributes 2* dataset [1]. We can easily infer that the word representation of "gorilla" and "chimpanzee" (at bottom right corner) are close to each other because they are semantically related. However, the representations of the words "gorilla" and "seal" are far apart due to their semantic dissimilarities.

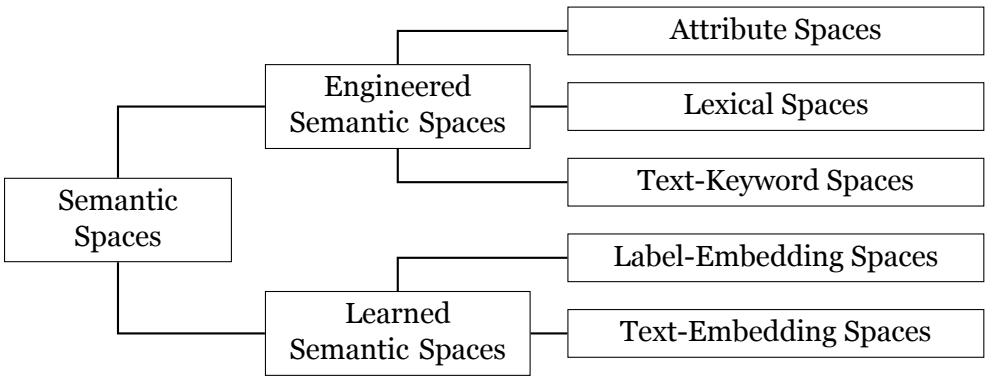


Figure 2.2: Classification of semantic spaces.

2.1.1.1 Attribute Spaces

Attribute spaces are composed by a set of *attributes*, i.e., a list of terms describing the class properties, denoted as attributes, and they are the most widely used semantic spaces in

zero-shot learning. Each attribute can be a word or a phrase, and these can be represented as real values (continuous attribute space) or binary values (binary attribute spaces) [15]. It can also be expressed as a measure that relates the relative degree of having an attribute among different classes (relative attribute spaces). For example, in the context of animal recognition problem, a zebra can be characterized by the color of body (e.g. "black", "brown", "yellow"), the habitat (e.g. "desert", "forest") or other attribute properties, like shape, part or material attributes [18]. For real value attributes, each of the properties can be expressed by the strength/confidence-level of having this attribute, e.g., -1 if the attribute is not present, 1 if the attribute is strongly noticeable, and a value between $[-1, 1]$ chosen if a specific attribute is slightly present. For binary attributes, each value denotes the presence or absence of a specific attribute.

2.1.1.2 Lexical Spaces

Lexical spaces are formed by a set of lexical items. This kind of spaces is based on the labels of the classes that can provide semantic information [15]. An example of a dataset that can be used to construct such spaces is the structured lexical database WordNet [19], in which the words are grouped together based on their meaning. The class prototype for the class c_i is a vector in which each dimension corresponds to one class, and the j-dimension represents the relation between class c_i and class c_j. Along with the structured databases, the datasets could also be some corpora. For example, each class can be represented as a co-occurrence vector of the class label with the N most frequent words from the corpus.

2.1.1.3 Text-keyword Spaces

Text-keyword spaces are semantic spaces composed of a set of text-keywords extracted from textual descriptions of each class, with each dimension corresponding to a keyword. In this kind of spaces, the most common source of information is websites, like Wikipedia [20]. For example, if we are interested in the task of recognizing flowers from their textual descriptions, we can extract a set of text-keywords from Wikipedia by searching for the name of the respective class. Then, the class prototypes can be constructed by the n keywords extracted from the Wikipedia. Some approaches are used to construct the text-keyword spaces, for example, [20] uses Bag-of-Words (BoW) representation for each text description.

2.1.2 Learned Semantic Spaces

Learned semantic spaces are not produced by humans, instead the class prototypes are obtained through machine learning models. The kind of models that are used to extract the prototypes for each class can be pre-trained models in other problems or trained particularly for the task of zero-shot learning. In the following sections, we introduce the learned semantic spaces used in zero-shot learning.

2.1.2.1 Label-Embedding Spaces

In the label-embedding spaces, the prototypes of each class are obtained through the embedding of class labels. In word embedding, the words are embedded into a space defined in $\mathbb{R}$ as vectors. The embedding space contains semantic information in which the embedded word vectors are close to each other if they are semantically related or far apart if they are semantically dissimilar. In existing works, different approaches have been adopted to the embedding task, such as Word2Vec [21] and GloVe [22]. As a footnote, both Word2Vec and GloVe take as input a text corpus and produce the word vectors as output. First, a vocabulary from the training text data is constructed and then the word representation vector is learned.

2.1.2.2 Text-Embedding Spaces

Similar to the label-embedding spaces, but make use of other type of embedding, in the text-embedding spaces the class prototypes are obtained by embedding the text descriptions for each class. The semantic information in these spaces comes from text descriptions like in text-keyword spaces but with some major differences. A text-keyword space is comprised through extracting keywords and using each of them as a dimension in the constructed space, whereas a text-embedding space is formed through some learning models. The text descriptions of each class are used as input of a suitable model, and the outputs are regarded as the prototypes of this class. For example, in the context of image object recognition, if the dataset provides the text descriptions for each sample, these ones are fed into a model and the output vectors of that model are the class prototypes.

2.2 Methods

We can classify existing zero-shot learning methods into three categories: (1) attribute classifier-based methods; (2) projection methods and (3) generative methods [15].
The attribute classifier-based methods aim to predict the attributes of an input and then infer the class label by searching the class that attains the most similar set of attributes. The projection methods focus on how to obtain labeled instances for the unseen classes by projecting the instances of the feature space or the instances of the semantic space into a projection space, which can be the semantic space or the feature space. More recently, the problem of zero-shot learning has been converted into a data missing problem, in which the goal is to generate some synthetic data of unknown classes using their semantic information [1]. Then, these generated data can be used to train a generic classifier, converting the zero-shot problem in a common classification problem. Therefore, generative models aim to learn a generator that can be used to randomly sample data from it.

Before describing each method, it is important to standardize the key notation used throughout this document. We summarize the key notations in Table 2.1.

Table 2.1: Key notations used in this document.

Notation	Description
X	Feature space, which is d-dimensional
X_s	Set of images belonging to seen classes, available during training
X_u	Set of images belonging to unseen classes, available during test stage
S	Semantic space, which is m-dimensional
A	Attribute space, which is m-dimensional
Y_s	Set of seen classes
Y_u	Set of unseen classes

2.2.1 Attribute Classifier-Based Methods

Early works of zero-shot learning [23] make use of a two-stage approach to infer the label of a given image. First, the attributes of a given image are estimated, and then the class label is inferred by determining the class that attains the most similar set of estimated attributes. In the following two sections, we describe two attribute classifier-based methods for being the most widely used methods in the literature [10].

2.2.1.1 Direct Attribute Prediction

Direct Attribute Prediction (DAP) was introduced by Lampert et al. in [23] to tackle the zero-shot learning problem. They propose the attribute-based classification that allows object detection based on a high-level description of these objects, called *attributes*, instead of making use of training samples. The semantic space used in this work falls into the category of attribute spaces, described in section 2.1.1.1. Specifically, in this work they only consider binary attributes. Figure 2.3 shows examples of per-class attribute annotation.

To make use of such attributes, the aim is to learn a classifier $\alpha : X \rightarrow Y_u$ if we dispose of an attribute representation $a^u, a^s \in A$ for each class $y_u \in Y_u$ and $y_s \in Y_s$, transferring information between Y_s and Y_u through A.

DAP [23] is illustrated in Figure 2.4. During training, the per-attribute parameters β_m are learned, using any supervised learning method. At test time, with the learned probabilistic attribute classifiers a class prediction is performed by combining scores of the learned attribute classifiers. The class label that attains the most similar set of attributes is then assigned to the test image.

A novel image is assigned to one of the unknown classes using:

$$f(x) = \arg\max_{y_u} \prod_{m=1}^{M} \frac{p(a_m^{y_u}|x)}{p(a_m^{y_u})} \quad (2.1)$$

where M is the number of attributes, $a_m^{y_u}$ the m-th attribute of class y_u, $p(a_m^{y_u}|x)$ is the attribute probability given the image feature x, which is obtained from the attribute classifiers. The $p(a_m^{y_u})$ is the attribute prior computed by the empirical mean of attributes over training classes.

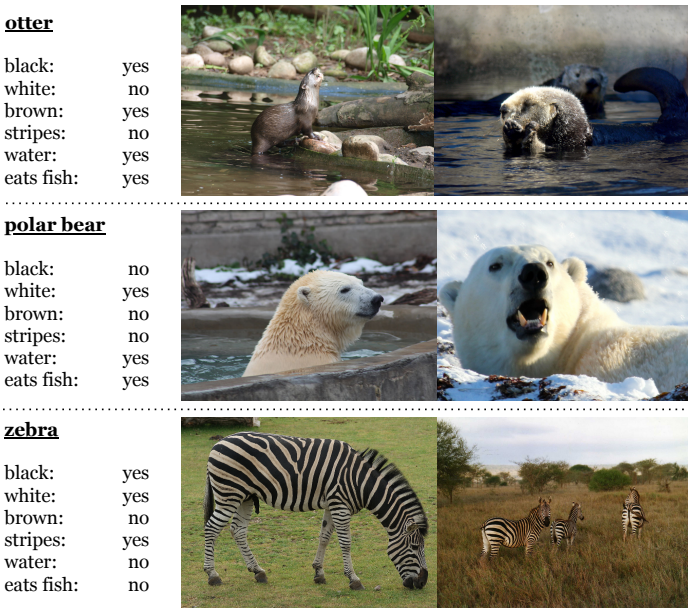


Figure 2.3: **Examples from *Animals with Attributes 2* [2] dataset of object classes with per-class attribute annotation**. The presence ("yes") or absence ("no") of an attribute is denoted respectively by "0" and "1" in the binary attribute vectors.

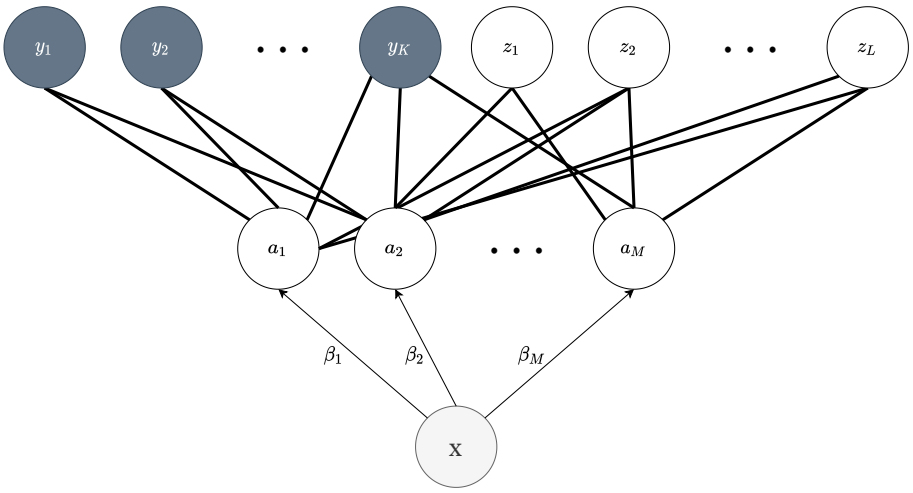


Figure 2.4: **Direct Attribute Prediction (DAP)**. During training, given the image features x of the training instances, the per-attribute parameters β_m are learned. At test time, with the learned probabilistic attribute classifiers, a class prediction is performed by combining scores of the learned attribute classifiers that establish a relationship with the seen (y) and unseen (z) classes. The class label that attains the most similar set of attributes is then assigned to the test image.

2.2.1.2 Indirect Attribute Prediction

Indirect Attribute Prediction (IAP) also uses the attributes to transfer knowledge between classes. In contrast to DAP, the attribute layer is placed between two label layers: the layer of known classes and the layer of unknown classes. IAP is depicted in Figure 2.5. During training, the aim is to learn a classifier for each training class, that is, predicting the probabilities of each training class given an image. In this way, IAP indirectly estimates

the probabilities of attributes for a given image x by the following equation:

$$p(a_m|x) = \sum_{k=1}^{K} p(a_m|y_k)p(y_k|x) \quad (2.2)$$

where K is the number of training classes, $p(a_m|y_k)$ is the probability of such class attribute given its class and $p(y_k|x)$ is the training class posterior obtained from the multi-class classifier. With the estimate of $p(a_m|x)$ obtained from Equation 2.2, we can predict the class label in the same way as in for DAP using the Equation 2.1.

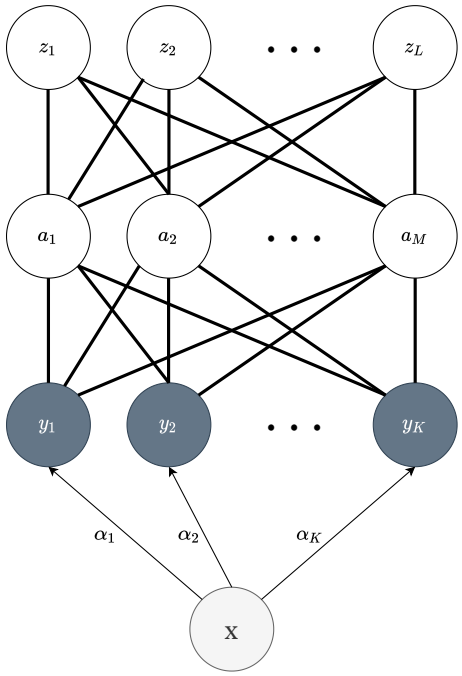


Figure 2.5: **Indirect Attribute Prediction (IAP)**. The parameters α_k are learned for each training class. At test time, the posterior distribution of the training class labels induces a distribution over the unseen class labels through the class-attribute relationship.

2.2.2 Projection Methods

The insight of projection methods is to learn a projection function that is used later to map the image features and the semantic attributes (class prototypes) into a projection/embedding space, which can be the feature space or semantic space. In the training phase, the projection function can be learned using an iterative method or, in some cases, using a direct method when the problem is cast as a system of linear equations [8]. At test phase, the classification is performed in the projection space using a distance metric method, to assign the closest class label to the projected image as the class prediction, or simple measuring the compatibility between the image feature and the correspondent class embedding vector. The class that attains greater compatibility value is then assigned to the test image. The figure 2.6 shows the block diagram of a common projection-based zero-shot method.

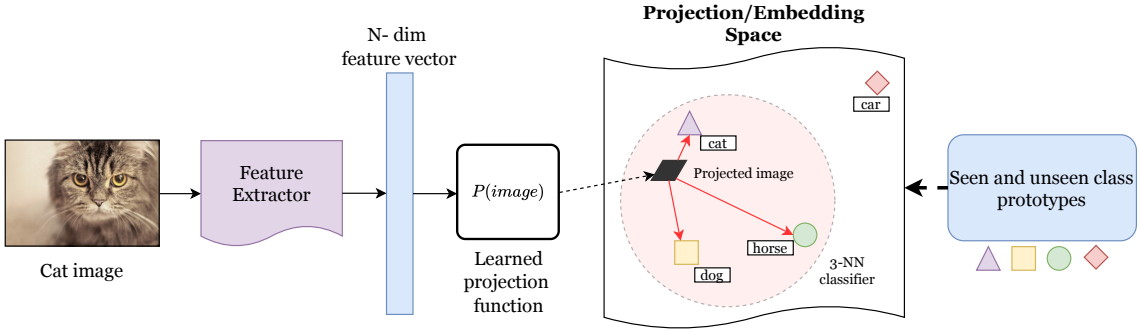


Figure 2.6: **Zero-shot learning using Projection-based methods**. The pipeline for this type of method is as follows: given an image from the test set, the feature representation vector is obtained through any feature extraction method. It is then projected into a projection space using the learned projection function. The classification is then performed in the projection space by a distance metric method or a *nearest neighbor classifier* (k-NN).

The feature space contains the training image features belonging to the seen class data and also the test image features, which belong to the unseen classes. The semantic space includes the class prototypes of both seen and unseen classes. Both spaces are defined in $\mathbb{R}$ with instances and class prototypes as vectors. As we can notice by figure 2.6, the projected feature vector (black parallelogram in figure) is nearest to the "cat" class prototype than others. Then, by performing the k-NN classification, the "cat" label is the class that will be assigned to the image.

In the following sections, we describe several state-of-the-art projection methods.

2.2.2.1 Semantic Output Codes

The Semantic Output Codes (SOC) classifier was proposed by Palatucci et al. [24] for the neuronal decoding task, i.e., the ability for predicting words that people are thinking based on the analysis of functional magnetic resonance images (fMRI), without having training examples for those words. To this end, SOC classifier uses the knowledge provided by the semantic properties of the unseen classes to transfer to novel classes.

Briefly, the SOC classifier is a two-stage process: (1) the training phase consists of learning the matrix W, which is used to map from the feature space to the semantic space; and (2) for the test phase, a *1NN* classifier is used. In other words, it returns the nearest point to the projected point in the semantic space according to the Euclidean distance.

To learn the mapping between the feature space and semantic space, the model uses a regression function, which is defined as follows:

$$\hat{W} = (X^T X + \lambda I)^{-1} X^T Y \quad (2.3)$$

where I is the identity matrix and λ a regularization parameter. Formally, given a novel fMRI image $x \in X_u$, the semantic features prediction $\hat{s}$ for this image is the result of multiplication of the image feature x by the learned $\hat{W} : \hat{s} = x.\hat{W}$. Then, a *1NN* classifier will return the closest point to the estimated semantic feature $\hat{s}$ in the semantic space, which is then mapped to the correspondent class label.

2.2.2.2 Cross-Modal Transfer

The Cross-Model Transfer (CMT) zero-shot model [3] implements a two-layer neuronal network to learn the non-linear mapping of image features to semantic space. It then uses a novelty detection mechanism to determine whether the test image belongs to seen or an unseen class. Thus, if the image belongs to the seen classes, then it is classified using a probabilistic classifier; otherwise, it is assigned to the class represented by the closest semantic word vector of the unseen classes.

During training, a set of training images $x^{(i)} \in X_s$ is mapped to a lower-dimensional space (semantic word space). The mapped image is represented by the word vector w_y. To learn the mapping of images into semantic word space, the following objective function needs to be minimized:

$$J(\Theta) = \sum_{y \in Y_s} \sum_{x^{(i)} \in X_s} \left|\left| w_y - \theta^{(2)} f(\theta^{(1)} x^{(i)}) \right|\right|^2 \quad (2.4)$$

where $f = tanh$ and $\Theta = (\theta^{(1)}, \theta^{(2)})$ are the parameters to be learned.

Due to the inability of standard classifiers predict a class that has no training examples, Socher et al. [3] propose a novelty detection strategy for predicting whether an image is of a seen or unseen class. Depending on the output of this detection method, each type of image can then be classified in two manners: (1) using a state-of-the-art softmax classifier, in case of the novelty detector predicts that the image is part of seen classes; or (2) using a Gaussian discriminator that compares the distances between word vectors of unseen classes and the word vector of the mapped image in the semantic space. Then, the predicted class is the one that is closest to the mapped image. Figure 2.7 illustrates the CMT model.

2.2.2.3 Deep Visual-Semantic Embedding Model

Frome et al. [4] present the *Deep Visual-Semantic Embedding* (DeViSE) model that learns semantic relationships between labels based on the textual data and adopt the semantic space as space where the classification is performed. Given a test image, the predicted class is the one with the highest compatibility with this image.
Conventionally, DeViSE model is a fusion of two modules: (1) a pre-trained neural lan-

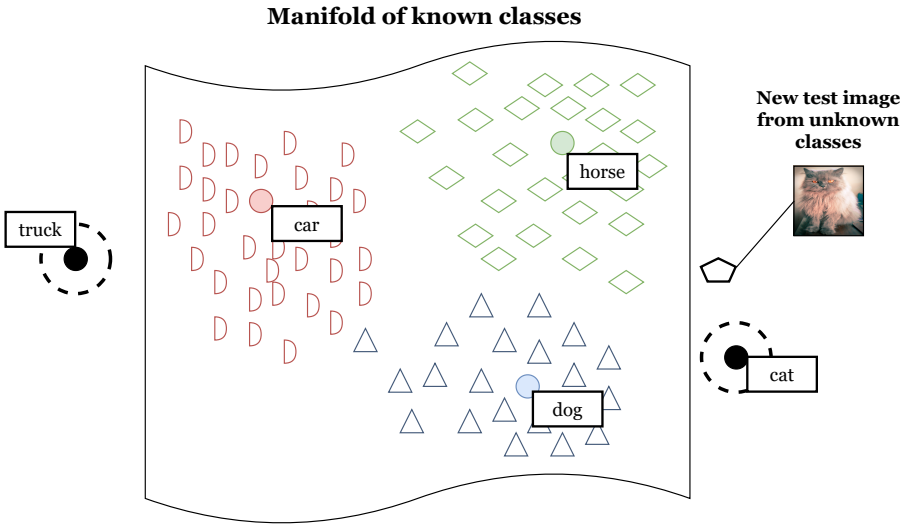


Figure 2.7: **Overview of CMT model** [3]. Given the test image from the unknown classes, the novelty detection method will determine whether the image belongs to seen or an unseen class. Then, the image will be classified according to the result of the novelty detector. If the image is "novel", it will be classified with the help of the other unknown classes; otherwise, the known classes are taken into account to perform the classification.

guage model and (2) a pre-trained visual model. For this reason, we can say that DeViSE is a hybrid model. Figure 2.8 illustrates the architecture of DeViSE model.

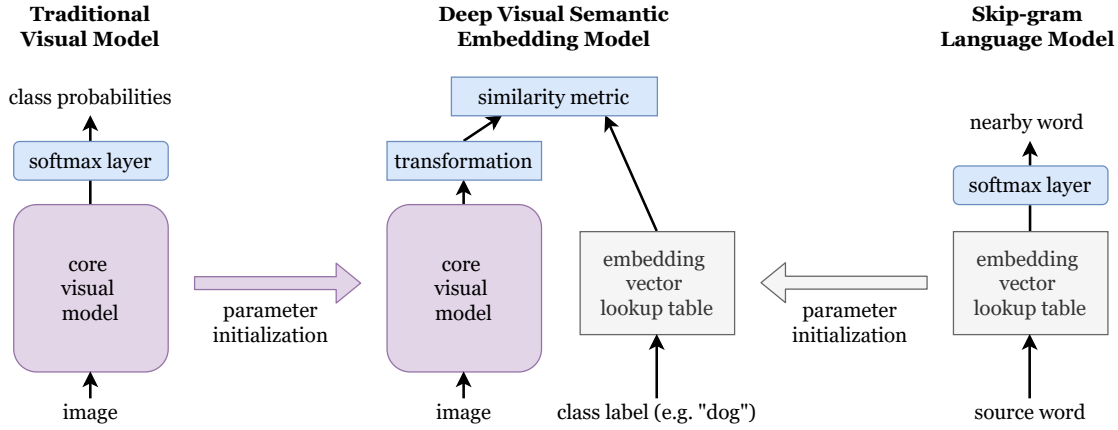


Figure 2.8: **Overview of DeViSE model** [4]. The visual model (left) is trained to produce the class probabilities of a given image. The skip-gram language model (right) is trained to learn the vector representation for each input word. Then, DeViSE model (center) is initialized with these two pre-trained models. Given a test image, the visual model will calculate its vector representation using the transformation layer and then the classification is performed by searching for the nearest labels in the embedding vector space.

The language model aims to learn the vector representation for each term (word) of textual data. A skip-gram text language model architecture is used due to its effectiveness in learning meaningful representations (embedding vectors) of terms from unannotated text [25, 21].

The visual model architecture is based on the winning model for the 1000-class ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 [26], since the ILSVRC 1K 2012 dataset [13] was used to pre-train the model. DeViSE is then initialized with these two

pre-trained neural networks models. The output embedding vectors produced by the language model are used to map label terms into target vector representations. The authors of [4] replaced the softmax layer of the visual model by a custom linear layer ("transformation" layer in figure 2.8), which maps the output of the last layer to a lower-dimensional output, enabling the comparison with the target vector (embedding vector) from the language model.

The chosen loss function is a fusion of dot-product similarity with hinge rank loss, such that the model is trained to produce a greater similarity between the output of the visual model and the correct vector representation of the image label than between the output of the visual model and other randomly text terms [4].

$$L(x, y) = \sum_{j \neq y} max[0, margin - s_y W\theta(x) + s_j W\theta(x)] \quad (2.5)$$

where $\theta(x)$ is the output of the top layer of the visual network for the given image x, W is the matrix of parameters in the transformation layer, s_y is the embedding vector for the provided text label, and s_j are the embedding vectors of other text terms. A ranking loss was used instead of an $L2$ loss because the evaluation of the nearest neighbor is mainly a ranking problem [4].

In the test phase, when a new image is presented, its vector representation is computed using the visual model; afterwards, the classification is achieved by searching for the nearest labels in the embedding space. The k closest labels are then assigned to the image. To measure the quality of predicted labels, an evaluation metric is used.

For the evaluation phase, Frome et al. [4] used the "flat" hit@k metric - a generalization of the top-1 accuracy with the difference that "flat" considers the top-k predictions. On the other hand, to measure the quality of predictions, in terms of semantic, the authors employed a hierarchical precision@k metric which is based on the label hierarchy presented in the ImageNet dataset [13]. Further on section 2.5, we describe these evaluation metrics in a more detailed manner.

2.2.2.4 Convex Combination of Semantic Embeddings

Convex Combination of Semantic Embeddings (ConSE) [27] is a simple method based on DeViSE [4], with the difference that the way to map an image into the semantic embedding space is done through convex combinations of the class label embedding vectors.

Definition 1. *A point $x \in R^n$ is a convex combination of the points $x_1, x_2, ..., x_r$ in R^n if for some real number $c_1, c_2, ..., c_r$ which satisfy $\sum_{i=1}^{r} c_i = 1$ and $c_i \geq 0$, $1 \leq i \leq r$, $x = \sum_{i=1}^{r} c_i x_i$. The convex combination operator is denoted by "$\cdot$".*

From the definition 1, we deduce that a convex combination is a fundamentally a linear combination of points (which can be vectors) where all coefficients are non-negative and its sum total 1 [28].

In the training phase, a softmax classifier is trained to estimates the probability of an image x belongs to a class label $y \in Y_s$, that is, $p_0(y|x)$, where $\sum_{y=1}^{n_0} p_0(y|x) = 1$. As well, a skip-gram model was trained on a corpus of billion words extracted from Wikipedia.org to create the word embedding vectors that define the semantic space, similar to DeViSE [4]. The objective is then to transfer the probabilistic predictions of the classifier to a set of test labels.

In the test phase, given a test image, the softmax classifier will return the T top predictions of the model (probability values). Then, the convex combination of the corresponding T embedding semantic vectors is calculated and the semantic vector $f(x)$ for the test image x is estimated. Formally,

$$f(x) = \frac{1}{Z} \sum_{t=1}^{T} p(\hat{y}_0(x,t)|x) \cdot s(\hat{y}_0(x,t)), \quad (2.6)$$

where Z represents a normalization factor T is a hyper-parameter that controls the number of embedding vectors to be considered, $p(\hat{y}_0(x,t)|x)$ denotes the probabilistic predictions given by softmax classifier and $s(\hat{y}_0(x,t))$ is the semantic representation vector for each seen class label.

Given the predicted semantic embedding vector in the semantic space for an image x, *i.e.*, $f(x)$, the zero-shot classification is performed by finding the class labels with closest embeddings to $f(x)$ in the semantic space. The top-1 prediction of the model for an image x from the test label set, denoted by $\hat{y}_1(x,1)$, is given by

$$\hat{y}_1(x,1) = \arg\max_{y' \in Y_1} cos(f(x), s(y')) \quad (2.7)$$

where $cos(.)$ denotes the cosine similarity metric, defined by $cos(f(x), s(y')) = \frac{f(x) \cdot s(y')}{\|f(x)\| \|s(y')\|}$ [29]. The cosine similarity was chosen as similarity metric to measure closeness in the semantic space, which means that a greater value of cosine similarity indicates a higher similarity between two vectors.

As an example, the figure 2.9 illustrates the overview of the ConSE method in predicting the class label for a test image.
In the same manner as DeViSE, the authors of [27] make use of "flat" hit@k and "hierarchical" precision@k evaluation metrics to measure the quality of predictions provided by the ConSE method.

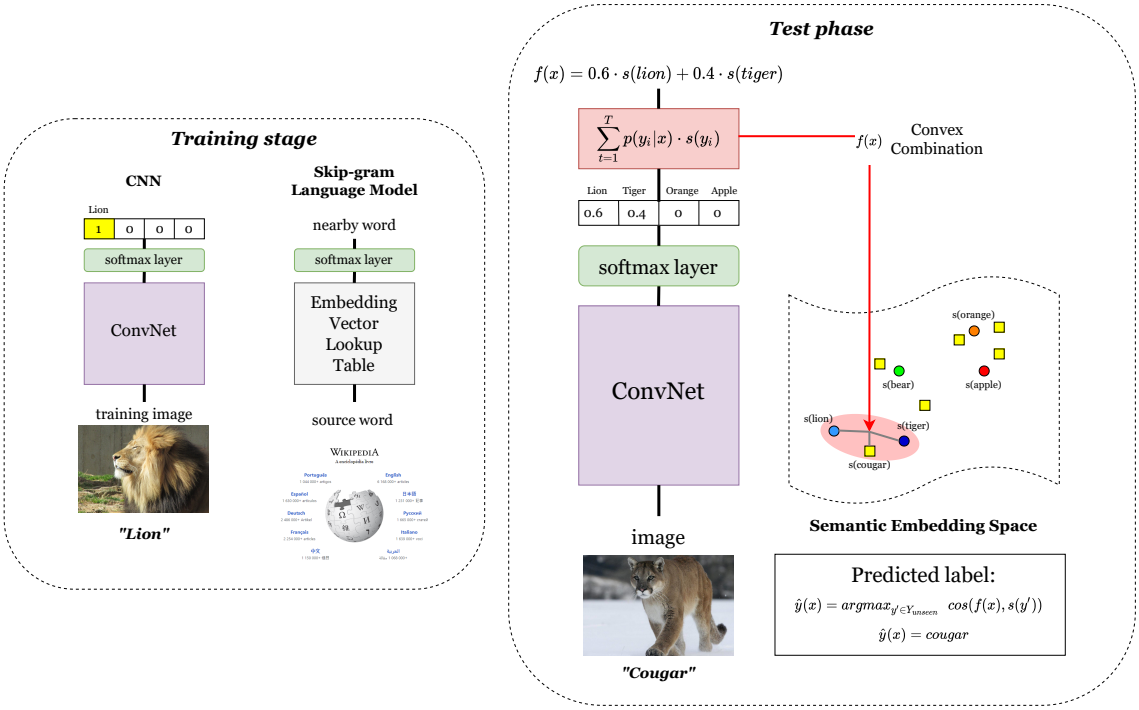


Figure 2.9: **Overview of ConSE model**. The test image "Cougar" is fed into a CNN, then the softmax top-layer returns the top T probabilistic predictions. Given these predictions, the semantic vector $f(x)$ for the test image x is estimated, by using equation 2.6. The semantic vector will be "placed" something between lion and tiger in the semantic space. Finally, the predicted label "Cougar" is obtained by the equation 2.7, $\hat{y}(x, 1) = \arg\max_{y' \in Y_u} cos(f(x), s(y'))$, which returns the closest test label.

2.2.2.5 Attribute Label Embedding

Attribute Label Embedding (ALE) [5] is an approach in which the class labels are incorporated into an attribute vector space. Akata et al. [5] propose a function to evaluate the compatibility between an image x and a label y. The parameters of this compatibility function are learned to guarantee that the correct class returns a higher compatibility value than the incorrect one, when an image is presented. Given a test image, classification consists of searching for the class which reaches the highest compatibility value with the presented image.

During training, the aim is to learn a matrix W that bridges between the image features and the attributes. For this purpose, is defined a compatibility function F, formally:

$$F(x, y; W) = \theta(x)' W \varphi(y) \quad (2.8)$$

where $\theta(x)$ is the visual feature vector of image x and $\varphi(y)$ is the attribute embedding vector in the attribute vector space. The objective function used in ALE is very similar to the unregularised structured SVM (SSVM) objective [30]:

$$L = \frac{1}{N}\sum_{n=1}^{N} \max_{y \in Y} \Delta(y_n, y) + F(x_n, y; W) - F(x_n, y_n; W) \quad (2.9)$$

where $F(.)$ is the compatibility function and W is the model parameter to be learned. At test stage, given a test image x, the predicted class label y is obtained by:

$$f(x; w) = \arg\max_{y \in Y} F(x, y; w) \quad (2.10)$$

where w is the model parameter learned during training and $F(x, y; w)$ measures how compatible is x and y given w. Thus, the predicted class is the one with the highest compatibility with the presented image x. We illustrate the ALE model in Figure 2.10.

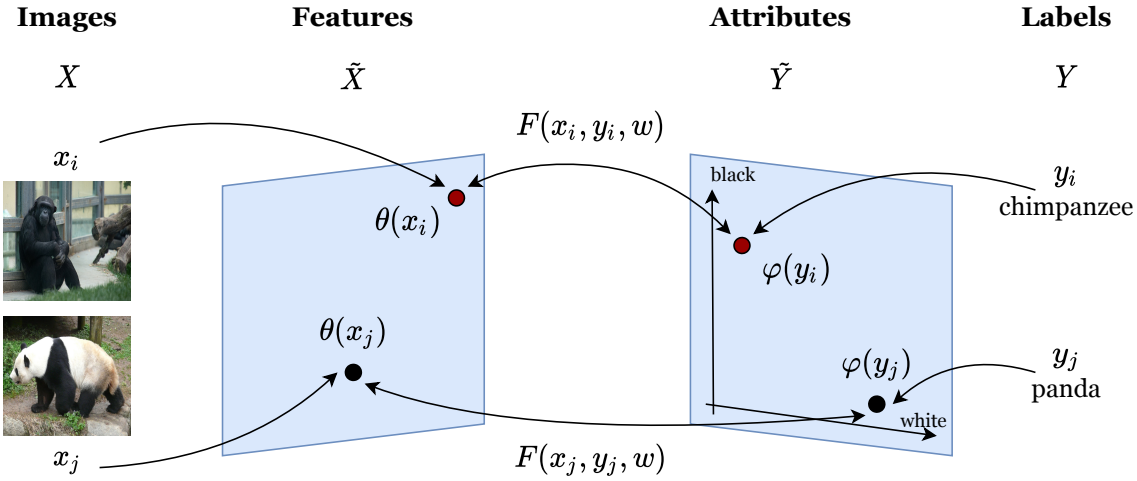


Figure 2.10: **Attribute Label Embedding model**. The idea is to leverage the attributes as side information for the label embedding and then measure the compatibility between the inputs (image feature) and outputs (class label) with the equation 2.10. It is desired that there is a higher compatibility between the feature vector $\theta(x_i)$, which represents a "chimpanzee", and its respective attribute vector $\varphi(y_i)$. The same logic is applied to the "panda" example. Adapted from [5].

2.2.2.6 Embarrassingly Simple Approach to Zero-Shot Learning

Embarrassingly Simple Approach to Zero-Shot Learning (ESZSL) [6] is a two-layer approach to model the relationship between features, attributes, and classes, with the help of a linear model. This approach follows the same guidelines as ALE [5], however, with a different loss function and regularizer, which makes the whole process simpler and efficient, leading to better results.

The training stage consists of learning the weights that describe the relationship between the image features and the attributes.
The second layer allows establishing the connection between the attributes and the classes using the final linear model obtained from the learned weights and the attributes.

At training stage, to learn the matrix V, which relates the features with the attributes, the authors of ESZSL consider the following problem:

$$minimize_{V \in \mathbb{R}^{d \times a}} L(X^T V S, Y) + \Omega(V) \quad (2.11)$$

where $W = VS$, L is the loss function and Ω a regularizer.

To solve the problem, Romera Paredes et al. [6] introduce a better regularizer and optimize a close form solution objective function in a bi-linear manner:

$$\Omega(V; S, X) = \gamma \left\| VS \right\|_{Fro}^2 + \lambda \left\| X^T V \right\|_{Fro}^2 + \beta \left\| V \right\|_{Fro}^2 \quad (2.12)$$

where γ, λ and β are the hyper-parameters of this regularizer, and $\|.\|_{Fro}^2$ denotes the Frobenius norm. The main benefit of this approach is that the objective function is convex and has a closed form solution [6]:

$$V = (XX^T + \gamma I)^{-1} XYS^T (SS^T + \lambda I)^{-1} \quad (2.13)$$

After learning the matrix V, at inference stage, the aim is to predict an unseen class label $\hat{y}$ by using its semantic attributes S_u weighted by matrix V. Then, given a new test image x', the predicted label is given by:

$$\hat{y} = \arg\max_i x'^T V S'_i \quad (2.14)$$

where VS' denotes the final linear model $W' = VS'$. To better visualize the problem, figure 2.11 summarizes the ESZSL framework.

The main advantages of this approach are that it can be implemented with one line of code, and is also computational efficient at both training and inference stages, due to the closed-form solution.

2.2.2.7 Latent Embeddings for Zero-Shot Learning

Another compatibility-based method is the Latent Embedding for Zero-Shot Classification (LatEm) [7], which is fundamentally a multi-modal method that learns several linear compatibility models in order to maximize the compatibility between an image and the respective class label.

During training, a bi-linear compatibility function is learned such that the images belong-

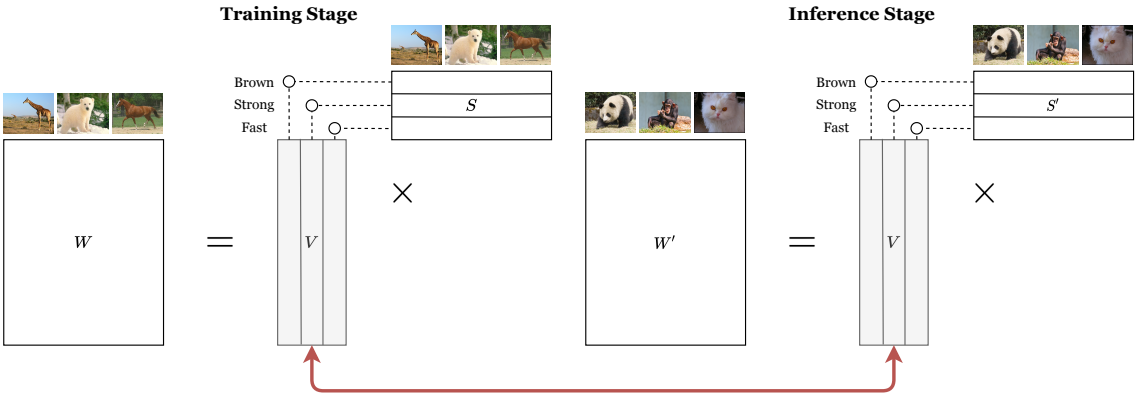


Figure 2.11: **Summary of ESZSL framework**. There are z_s classes in the training stage, each one described by a_s attributes. At the training stage, the matrix S, denoting the semantic attributes, is used together with the training instances to learn the matrix V, which corresponds to mapping from feature space to attribute space. On the other hand, at the inference stage, the matrix V is used jointly with the semantic attributes of test classes, S', to obtain the final linear model $W' = VS'$. The predicted label is given by $\hat{y} = \arg\max_i x^T V S'_i$. Adapted from [6].

ing to the same class are all grouped together and the images of different classes are far away from each other. Once learned, the bi-linear function can be used to predict the class of a test image. For better visualization, figure 2.12 shows the overview of LatEm framework.

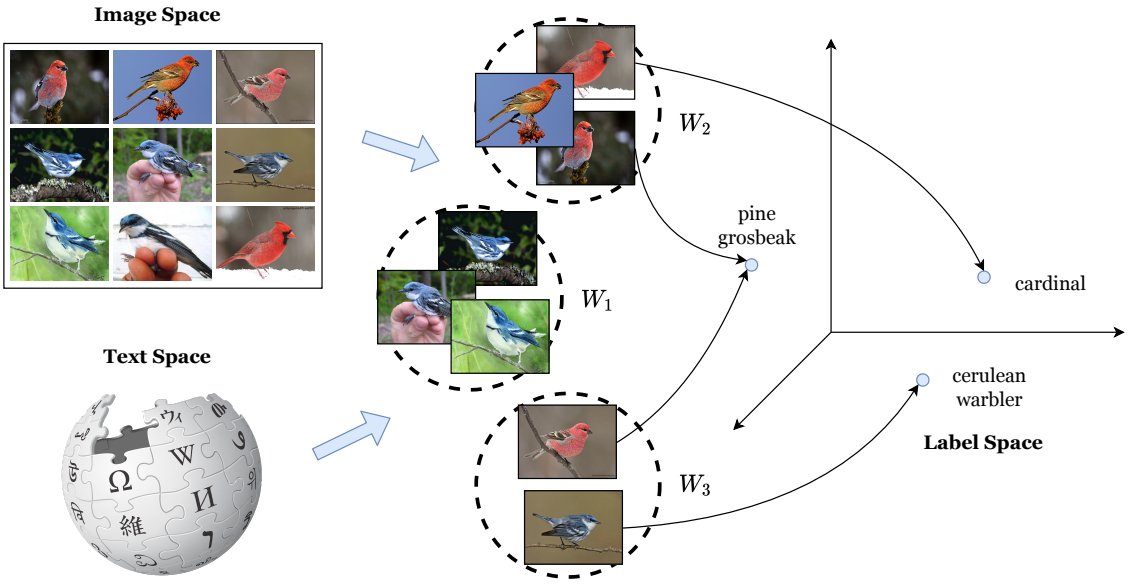


Figure 2.12: **Overview of LatEm framework**. The main purpose of LatEm is learning multiple models, W_i, towards maximizing the compatibility between the input embedding (image, text space) and the predicted class label. The various learned models, W_i, may capture distinct visual aspects of objects, enabling the model to enhance the classification accuracy. This way, the idea is to choose the model, W_i, that adequately classifies a given test example. Adapted from [7].

More formally, at training stage, the compatibility function takes the form:

$$F(x, y) = x^T W y \quad (2.15)$$

where W is the matrix intended to be learned, x the image feature, and y the class label. However, assuming that we want to learn multiple W_i, we need to rewrite the equation 2.15 as follows:

$$F(x, y) = \max_{1 \leq i \leq K} x^T W_i y \quad (2.16)$$

where $i = 1, ..., K$, with $K \geq 2$. To this end, the aim is to learn a set of compatibility spaces that minimizes the following loss function:

$$\frac{1}{N} \sum_{n=1}^{|T|} L(x_n, y_n) \quad (2.17)$$

where L is the ranking-based loss function, expressed as:

$$L(x_n, y_n) = \sum_{y \in Y} max\{0, \Delta(y_n, y) + F(x_n, y) - F(x_n, y_n)\} \quad (2.18)$$

where $\Delta(y_n, y) = 1$ if $y \neq y_n$ and 0 otherwise. Thus, the model is trained to returns a high compatibility value between an image embedding and the respective class embedding.
At the inference stage, once learned the models, W_i, zero-shot classification is performed by using the following prediction function that returns the class with the maximum compatibility:

$$f(x) = \arg\max_{y \in Y} F(x, y) \;\; s.t. \;\; F(x, y) = \max_{1 \leq i \leq K} x^T W_i y \quad (2.19)$$

with $y \in Y$ being one of the unseen classes.

2.2.2.8 Semantic Autoencoder

Kodirov et al. [8] proposed a model based on the learning of a semantic autoencoder (SAE). According to the encoder-decoder paradigm, the encoder aims to project a visual feature vector into the semantic space. The decoder comprises an additional constraint - it must be capable to reconstruct the original visual feature vector. This assumption makes the model more capable in generalizing to unknown classes. Since both the encoder and decoder are symmetric and linear models, they enable an extremely efficient learning algorithm with a low computational cost.

The proposed semantic autoencoder is illustrated in figure 2.13.

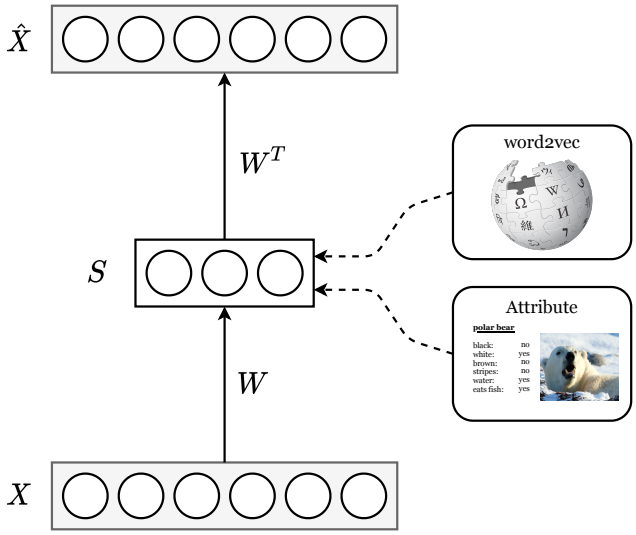


Figure 2.13: **Proposed Semantic Autoencoder for ZSL**.The encoder, W, aims to project a given input X to a hidden layer S (semantic space) that has a lower dimension, and the decoder, W^T, projects the learned projection back into the feature space, reconstructing the original feature $\hat{X}$. Then, the classification can be performed in feature space or in semantic space. Adapted from [8].

More formally, given an input data matrix X, each feature vector is projected into a latent representation S through the projection matrix W, that is, $S = XW$. The latent representation S is projected back into the feature space through the projection matrix W^T, and the result is the reconstructed input $\hat{X} = SW^T$. To this end, is desirable that $\hat{X}$ be as similar as possible to X, then the following objective function should be minimized:

$$\min_{W} \left\|X - W^T S\right\|_F^2 \quad s.t. \;\; WX = S \quad \textbf{(2.20)}$$

After few optimization techniques we fall in the following formulation:

$$AW + BW = C \quad \textbf{(2.21)}$$

with $A = SS^T$, $B = \lambda XX^T$, and $C = (1+\lambda)SX^T$, which is the formulation of the Sylvester equation and it can be solved by the Bartels-Stewart algorithm [31].

At the inference phase, given the semantic representation S and the training data X_s, the encoder W and decoder W^T are firstly learned by Bartels-Stewart algorithm. Afterward, zero-shot classification can be performed in two spaces:

1. **Semantic space** – With the encoder matrix W, the training example x_i is projected into the semantic space by $\hat{s}_i = Wx_i$. Then, the classification is performed by calculating the distance between the estimated semantic representation $\hat{s}$ and the

projected prototypes S_u:

$$\Phi(x_i) = \arg\min_j D(\hat{s}_i, S_{u_j}) \quad (2.22)$$

where D denotes for a distance function and $\Phi(.)$ returns the predicted class given the example x_i.

2. **Feature space** – With the decoder matrix W^T, the class prototypes s_i are projected into the feature space by $\hat{x_i} = W^T s_i$. After that, the classification can be performed in the feature space by calculating the distance between the estimated feature representation $\hat{x_i}$ and the projected prototypes X_u:

$$\Phi(x_i) = \arg\min_j D(\hat{x_i}, X_{u_j}) \quad (2.23)$$

where D denotes for a distance function and $\hat{X}_{u_j}$ is the j-th unseen class prototype projected in the feature space.

Despite this flexibility at test phase, the authors of SAE [8] reported that the two strategies described earlier yield very similar results.

2.2.3 Generative Methods

The main shortcoming with projection-based methods is that they suffer from two problems: bias and domain shift [8]. Due to the fact that the projection function is learned using solely seen classes during training, the projection method will be biased to predict seen class labels as the output owing to the imbalance between seen and unseen classes [1]. There is no guarantee that the learned projection function will map unseen class image features to the corresponding semantic space accurately at test time. This is because the learning method has only learned to map seen class image features to semantic space during training and might not generalize for unseen novel classes at test time. Thereby, most of existing state-of-the-art approaches fail to achieve reasonable results for the generalized zero-shot learning. To overcome this drawback, generative-based methods come into the picture. Generative methods aim to generate image features for unseen classes using any auxiliary information, such as semantic attributes or word vectors [1, 9]. Typically, this is achieved by using a conditional generative adversarial network [1, 10] or a conditional variational autoencoder [9, 10, 32] that leverages auxiliary information to generate image features for the unseen classes.
The figure 2.14 shows the diagram of a typical generative-based zero-shot method.
Similar to the projection-based methods, a feature extractor network is used to get a N-dimensional feature vector. First, any type of auxiliary information (e.g attributes, word vectors) is fed into the generative model, as shown in the figure 2.14. The generator module generates an N-dimensional output vector conditioned on the attribute vector (auxiliary information). The generative model is trained to synthesize feature vectors that

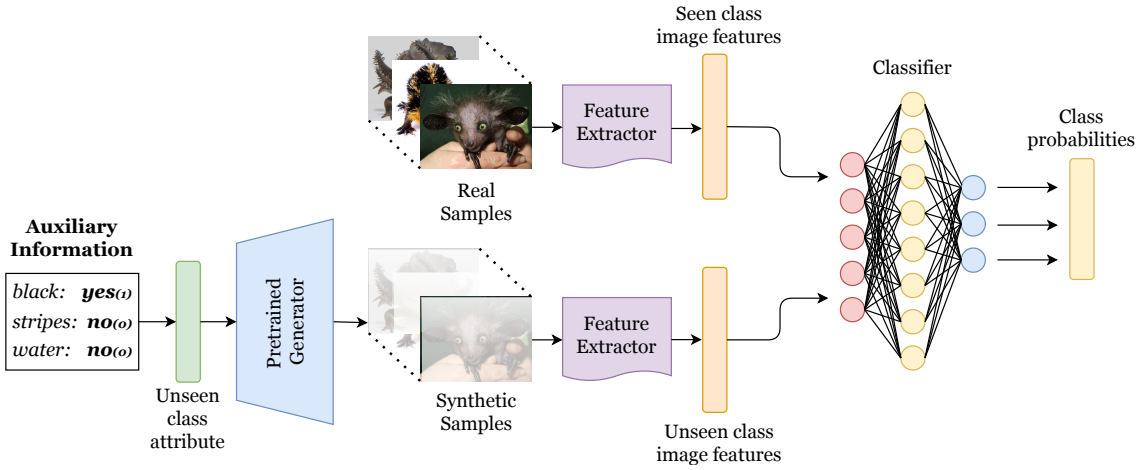


Figure 2.14: **Zero-shot learning using generative-based methods**. By generating some synthetic image features, belonging to unseen classes, conditioned on the side information (e.g. attributes), using a pre-trained generator, we can use these synthetic features to train a softmax classifier.

compared to the original N-dimensional feature vectors. Having this synthetic data, we can train a discriminative classifier, such as a softmax classifier, to predict the class labels of the test data. In other words, this family of approaches casts the zero-shot learning problem as a data missing problem and the view is to generate some synthetic data (data augmentation mechanism) to tackle this issue.

The main insight of this kind of methods is that by feeding additional synthetic features of unseen classes, the learned classifier will also explore the embedding space of unseen classes. Hence, the key is the ability to generate semantically meaningful features conditioned on a class specific semantic vector, without access to any images of that class.

In the following sections we describe some generative-based methods for zero-shot learning.

2.2.3.1 f-CLSWGAN

As stated above, most of existing projection-based methods fail to achieve reasonable results for the task of zero-shot classification due to the imbalance between seen and unseen classes, since they not have access to test classes during training. To surpass this issue, Xian et al. [1] propose a novel attribute conditional feature generating adversarial network (GAN) approach - namely f-CLSWGAN - that synthesizes features for the unseen classes conditioned on semantic information of unseen classes. By using a Wasserstein GAN with a classification loss, their approach is able to generate discriminative features to train a discriminative classifier to perform zero-shot classification. The f-CLSWGAN is depicted in figure 2.15.

During training, the generative network is trained to generate discriminative features with the help of the discriminator and the classification loss. Shortly, the objective function is expressed as:

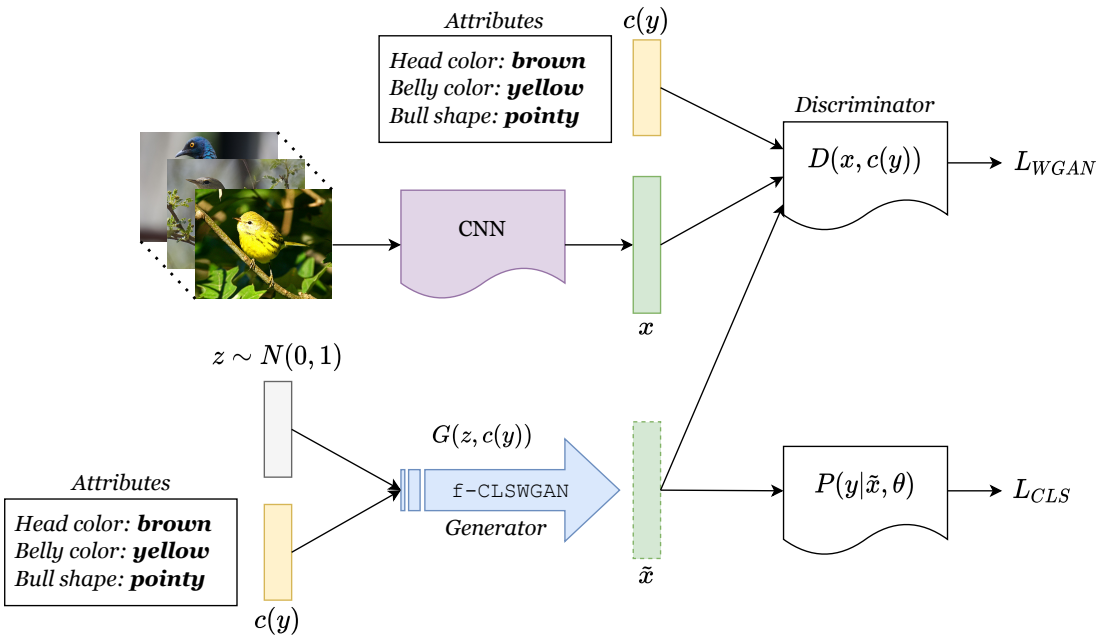


Figure 2.15: **f-CLSWGAN model**. The generator G aims to generate discriminative features conditioned on unseen class attributes, such that the generated features looks real for the discriminator D. When this assumption is achieved, the generated features can be used as unseen class data to train a discriminative classifier. The authors of f-CLSWGAN propose to minimize both the classification loss, L_{CLS}, over the generated features $\tilde{x}$ and the Wasserstein distance L_{WGAN}. Adapted from [1].

$$\min_G \max_D L_{WGAN} + \beta L_{CLS} \quad (2.24)$$

where β is a hyper-parameter that weights the classifier.

At the classification time, once we have features to all of classes, seen and unseen classes, we only have to train a softmax classifier in order to predict the class labels. More formally, the prediction function is written as follows:

$$f(x) = \arg\max_y P(y|x;\theta) \quad (2.25)$$

where in ZSL setting, $y \in Y^u$ and in GZSL setting, $y \in Y^s \cup Y^u$.

Additionally, with the generated features for the unseen classes, it is possible to train some methods described in the previous section, which learn a model to relate the image features with the class label, such as ALE [20], DEVISE [4], ESZSL [6] and LATEM [7]. However, the standard softmax classifier was the one that achieved better results in their experiments.

2.2.3.2 CVAE-ZSL

The idea behind the proposed approach - namely CVAE-ZSL - by Mishra et al. [9] is to generate samples for the unseen classes based on the given attributes using a Conditional Variational Autoencoder (CVAE).
A CVAE uses an additional input to both encoder and decoder that is called a condition [33]. This condition represents a property related to the example intended to be generated. For instance, if the purpose is to generate some images of dogs, a good condition is the set of attributes allowing generator synthesize images or features related to class "dog". Now, the idea is exactly to train a CVAE to learn a probability distribution of the image features X conditioned on the class embedding vector A. Furthermore, the authors of [9] pointed that such an approach reduces the domain shift problem.

Figure 2.16 illustrates the network architecture of CVAE-ZSL approach that is fundamentally the CVAE architecture.

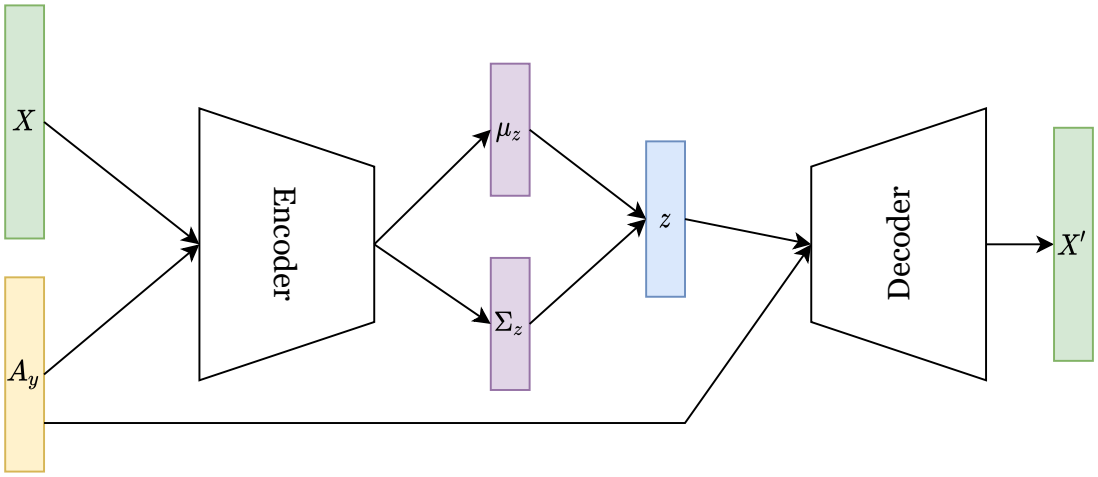


Figure 2.16: **CVAE-ZSL network architecture**. The input X, i.e., the image feature, and the semantic class embedding vector A_y are concatenated and passed through the encoder, composed of dense layers, that outputs a mean vector μ_z and a standard deviation vector Σ_z. A z is sampled from the variational distribution $N(\mu_z, \Sigma_z)$. The sampled z is then concatenated with the class embedding vector A_y and the result is passed through the decoder, which outputs the reconstructed image feature X'. Adapted from [9].

During training, CVAE generates image features $\tilde{x}$, given the conditional variable A_y. To this end, the loss function that is minimized is composed by a "reconstruction term" and a "regularization term", and it is defined as:

$$L(\theta, \phi; x, A_y) = L_{reconstruct}(x, \tilde{x}) + KL(N(\mu_z, \Sigma_z), N(0, 1)) \quad (2.26)$$

where KL denotes the Kullback-Leibler divergence, x is the input to the encoder and $\tilde{x}$ is the reconstruction output.

After training the CVAE, it is possible to generate examples for the unseen classes, since their semantic class embedding vectors are available. More formally, decoder operates as

a sample generator using the following procedure: 1) sample z from a standard normal $N(0, 1)$ and concatenate it with A_y, which is the attribute embedding; 2) the concatenation is used as the input to the decoder that outputs the reconstructed x; and 3) the generated samples can be used to train any classifier. In this work, Mishra et al. [9] trained an SVM classifier [34]. The pipeline of the zero-shot classification algorithm is the follow:

1. Using X_{train}, Y_{train} and A to train the CVAE.
2. For each unseen class y_u, generate N samples for that class, concatenating the latent vector $z \sim N(0, 1)$ and A_{y_u} and use it as the input to the decoder.
3. Train an SVM classifier using the generated samples for the unseen classes.

Depending on the setting where the model is evaluated, the test set contains only unseen classes (standard zero-shot learning) or both seen and unseen classes (generalized zero-shot learning).

2.2.3.3 f-VAEGAN-D2

The above two sections describe two generative methods that rely solely on GANs and VAEs. However, Xian et al. [10] developed a conditional generative model that combines the strength of VAE and GANs by bringing them together in a conditional feature generative model, namely f-VAEGAN-D2, which generates image features from class-level side information, such as attributes or word vectors. In addition, they employed an additional discriminator (D2) that distinguishes real and generated features from unlabeled data of unseen classes, which leads to more discriminative features. In zero-shot learning, the use of unlabeled test instances from unseen classes during training time is known as transductive setting, which inherently improves the model's performance, as they contain useful information of unseen classes [2]. However, most existing ZSL methods lay on the inductive setting, in which only labeled instances of seen classes are considered at the training phase [2].
The f-VAEGAN-D2 model consists of a conditional encoder, a shared decoder/generator, a conditional discriminator, and a non-conditional discriminator that grant to the model the ability to act in any-shot learning scenarios. Figure 2.17 illustrates the model. Once trained, the model generates discriminative image features for unseen classes that can be used to train a softmax classifier.
At training time, the generator $G(z, c)$ and the decoder $D(z, c)$ are trained in a way such that the generated features are discriminative enough to fool the discriminator $D_1(x, c)$. Moreover, if unlabeled data of novel classes are available, the non-conditional discriminator D_2 comes into play to distinguish between real and generated features of unseen classes.

Hence, f-VAEGAN-D2 model optimizes the following objective function:

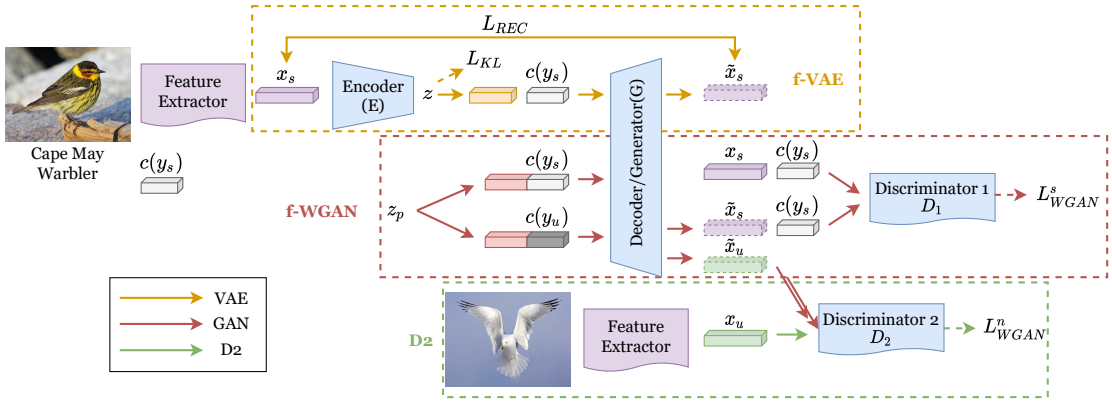


Figure 2.17: **Any-shot feature generating network (f-VAEGAN-D2)**. The presented architecture consists of a feature generating VAE (f-VAE), a feature generating WGAN (f-WGAN) with a conditional discriminator (D1), and a transductive feature generator with a non-conditional discriminator (D2) that learns from labeled data of seen classes jointly with unlabeled data of unseen classes. Both the feature generating VAE and WGAN are used to generate features from a random noise z and a condition c, such that the generator $G(z, c)$ of the GAN and decoder $D(z, c)$ of the VAE share the same parameters. Adapted from [10].

$$\min_{G,E} \max_{D_1,D_2} L^s_{VAEGAN} + L^n_{WGAN} \quad (2.27)$$

At test time, with the generated discriminative CNN features for the unseen classes, and in the same way as f-CLSWGAN [1] and CVAE-ZSL [9], we can simply train a softmax classifier to distinguish between seen and unseen classes.

To conclude, the major strength of this approach is the generation of semantically rich CNN features, generalized to any-shot learning scenarios. The results showed that this robust generative model improves the state-of-the-art in any-shot learning, i.e., inductive and transductive zero-shot learning.

2.2.3.4 Semantic-Guided Multi-Attention

All above methods ignore the importance of learning discriminative visual features, especially in the fine-grained scenarios. Usually, the features learned from the whole image using a common feature extractor hardly capture the subtle differences between classes. To capture such discriminative regions, Zhu et al. [35] propose a semantic-guided attention localization model (SGMA) for zero-shot recognition, which discovers the crucial regions.

The framework consists of three modules: (1) the multi-attention subnet, (2) the region cropping subnet, and (3) the joint feature embedding subnet. The multi-attention subnet aims to generate multiple attention maps that correspond to discriminative parts of the object. The region cropping subnet performs the cropping of these discriminative parts. The joint feature embedding subnet is fed by the cropped parts and the original image to learn the global and local visual feature.

The model is trained to produce attention maps that assemble multiple discriminative regions of the images. Then, the local features are obtained through the cropping of such regions. To perform the classification, the local features are concatenated with the global feature, and the result is used to rank the compatibility of the image feature with the true class label. The other way is to infer the class label based on the similarity between the final feature and the prototypes of unseen classes, similar to [7] and [5].
This method clearly shows the importance of capture discriminative parts of images to enhance the zero-shot recognition in fine-grained datasets.

2.3 Challenges

Despite the success of the above methods in addressing the zero-shot learning problem, some of them are faced with inherent issues that affect the performance of the model.
In the following sections, we describe some of the major challenges discovered in zero-shot learning that have a crucial impact in the model's performance.

2.3.1 Domain-Shift

The domain-shift problem was firstly identified by Fu et al. [36]. Domain-shift concerns the problem of having training and test data stemming from different distributions. This is a common problem in the projection-based methods since the projection functions learned from the training data are biased when applied directly to the test data, which contains disjoint classes, i.e. $Y_s \cap Y_u = \varnothing$ [36]. Since the projection function is learned using only seen classes during training, it might not generalize well on unseen classes at test time. To overcome this problem, a reconstruction constraint [8] and a data generation technique [9] proved to be efficient in reducing the domain-shift problem.

2.3.2 Bias

In the inductive zero-shot learning setting, the model has only access to images and respective class labels of seen classes and semantic class information of both seen and unseen classes during the training phase. This strong assumption makes the model biased to the seen classes in the test phase since it usually predicts the seen classes as the correct class. This problem becomes critical when the model's evaluation is under the generalized zero-shot learning setting, in which the search space contains both seen and unseen classes. In this way, the model tends to misclassify most unseen class images, significantly reducing classification accuracy [37]. Mishra et al. [9] used the data generated from both the seen and unseen classes to train a SVM classifier to reduce the bias towards the seen data.

2.3.3 Hubness

The problem of "hubness" emerges when a vector is projected from a high dimensional space into a low dimensional space [38]. As stated in [38], in projection methods, when

the semantic space is adopted as projection space, the instances of the feature space are projected into the semantic space. This operation can lead to some projected prototypes being nearest neighbors of a significant number of instances of different classes [15]. These prototypes are called "hubs", hence the name "hubness". It is known that the presence of hubs often hurts the classification performance. Then, to alleviate the hubness effect, some methods choose other projection space [8, 39], such as the feature space. In contrast, others consider using the unlabeled testing instances to increase the performance of the model [38].

2.4 Datasets

Among the most widely used datasets for zero-shot learning, we select the ones that were used to evaluate the above-reviewed methods, namely *Animals with Attributes 1* (AWA1) [23], *Animals with Attributes 2* (AWA2) [2], *Caltech-UCSD-Birds* (CUB-200-2011) [40], *Attribute Pascal and Yahoo* (aPascal-aYahoo) [18], *Stanford Dogs* [41], CIFAR-10 [42], CIFAR-100 [42], PubFig [43], OSR dataset [44] and ImageNet [13]. Due to copyright issues on the AWA1, the original images are not available, and thus, Xian et al. [2] introduced the AWA2 dataset, which has more 6847 images than the original AWA1. Details of dataset statistics are present in Table 4.2.

Table 2.2: Statistics for datasets.

Dataset	No. Classes	No. Instances	No. Attributes
CUB-200-2011 [40]	200	11,788	312
Oxford 102 Flower [45]	102	8,189	None
SUN Attibutes [46]	717	14,340	102
Stanford Dogs [41]	120	20,580	None
AWA [23]	50	30,475	85
AWA2 [2]	50	37,322	85
aPascal-aYahoo [18]	32	15,339	64
PubFig [43]	200	58,797	None
PubFig-sub [43]	8	772	11
OSR [44]	8	2,688	6
ImageNet [13]	22,000	15 million	None
ImageNet 2012 1K [47]	1,000	1.2 million	None
CIFAR 10 [42]	10	60,000	None
CIFAR 100 [42]	100	15,339	None

It is common to use a pre-trained deep neural network (DNN) for image feature extraction; however, the data used to train these DNN should not include any test classes used to evaluate the zero-shot learning model. Xian et al. [10] noticed that, from the standard split (SS) of aPascal-aYahoo and AWA1 datasets, more than half of test classes were among the classes used to pre-train the DNN, which violates the zero-shot assumption. In addition, the reported accuracy for all methods on the standard split (SS) is generally higher than the others. Therefore, Xian et al. [10] proposed new dataset splits, denoted proposed splits (PS), to ensure that none of the test classes appear in the dataset used to pre-train the DNN model, specifically ResNet, which is trained with ImageNet 1K [47]

dataset. Further on, in Section 2.6, we can compare the zero-shot learning results on SUN, CUB, AWA1, AWA2 and aPY using the standard split (SS) and the proposed split (PS). We also provide some reported results for the large-scale ImageNet dataset.

2.5 Evaluation Metrics

The published results by existing works are often not comparable and tend to be inaccurate due to some lack of care in maintaining the zero-shot assumption [2]. Therefore, Xian et al. [2] propose a unified evaluation protocol that aims to standardize the zero-shot learning benchmark. Typically, zero-shot learning methods are evaluated in two settings:

- **Zero-Shot Learning Setting** (ZSL) – It is the standard evaluation setting, in which at test time only unseen classes are present, i.e. $y \in Y_u$. However, this setting is considered restricted and somewhat unrealistic [2, 9].
- **Generalized Zero-Shot Learning Setting** (GZSL) – It is a more challenging and also a more realistic setting since, at test time, the search space contains both seen and unseen classes, i.e., $y \in Y_u \cup Y_s$ [48]. Consequently, the performance under this setting degrades significantly compared to conventional zero-shot learning. Since the model is trained only with seen class images, therefore its predictions are biased towards seen classes, leading to wrong classification of some unseen classes, which drastically reduce the performance [2].

In order to measure the performance of zero-shot learning methods using the above two settings (ZSL and GZSL), we provide a description of four evaluation metrics suitable for each type of setting.

2.5.1 Top-1 Accuracy

To measure the accuracy in image classification problems, top-1 accuracy is used, i.e., the prediction is correct when it corresponds to the true class label. This approach is encouraged if the classes are densely populated since the accuracy is averaged for all images. Conversely, if the dataset is imbalanced, which drives the classes being sparsely populated, the better approach is to measure the average per-class top-1 accuracy as follows:

$$acc_y = \frac{1}{|Y|} \sum_{c=1}^{|Y|} \frac{\#correct\ predictions\ in\ c}{\#samples\ in\ c} \quad (2.28)$$

where Y is the total number of classes. Xian et al. [2] observed that due to the class imbalance in the dataset AWA1, there is a significant difference (about 4%) in the top-1 accuracy per-image and the average per-class top-1 accuracy.

2.5.2 Harmonic Mean

In the GZSL setting, the search space contains both train and test classes, hence this setting is more realistic. Thus, after computing the average per-class top-1 accuracy on training and test classes, the harmonic mean [49] is computed for the training and test accuracies:

$$H = \frac{2 * acc_{y^{tr}} * acc_{y^{ts}}}{acc_{y^{tr}} + acc_{y^{ts}}} \quad (2.29)$$

where $acc_{y^{tr}}$ and $acc_{y^{ts}}$ denote the accuracy of images from seen and unseen classes, respectively. The choice of the harmonic mean is due to the fact that it provides more realistic results, since the arithmetic mean overestimates the real performance of the method [2].

2.5.3 Flat Hit@K

The flat hit@k metric is a generalization of the top-1 accuracy metric. The major difference is that flat hit@k considers the top-k predictions of the model compared to top-1 accuracy metric, which only takes into account the top-1 prediction [4]. This metric was used in [4] and [27] to measure the performance of the model on a test set taken from ImageNet ILSVRC 2012 1K, a large-scale dataset with 1000 classes. The results show that the larger k is, the better is the performance of the model.

2.5.4 Hierarchical Precision@K

To assess the semantic quality of predictions beyond the correct label, Frome et al. [4] employ a hierarchical precision@k metric, $hp@k$, to evaluate the accuracy of model predictions in relation to the object category hierarchy of the ImageNet dataset. For each image in the test set, the model returns its top k predicted labels of the ImageNet object category. The $hp@k$ is computed as the fraction of the top k predictions returned by the model, and that are part of the $hCorrectSet$, averaged across the total number of test samples [4]:

$$hp@k = \frac{1}{N} \sum_{i=1}^{N} \frac{\text{number of model's top } k \text{ predictions in } hCorrectSet \text{ for image } i}{k} \quad (2.30)$$

To construct the $hCorrectSet$, it is necessary adding nodes from the ImageNet hierarchy in a specific radius around the true label until $hCorrectSet$ has size $\geq k$.

2.6 Experiments

To compare the aforementioned methods, we provide both ZSL and GZSL results on the SUN [46], CUB [40], AWA1 [23], AWA2 [2], and aPY [18] datasets. Finally, we present

results on the large-scale ImageNet [13] dataset.

Table 2.3 presents the zero-shot learning results on standard split (SS) and proposed split (PS) for all above described methods, except for SOC [24] method, which does not report any values for the above mentioned datasets. Whenever available, the reported results are taken from the original publications. However, since not all methods provide results for all mentioned datasets, we fill the gaps with the results reported in [2]. Furthermore, the works [6] and [8] were re-evaluated using the proposed splits provided by [2] and code from original papers, when available. To re-evaluate the ESZSL [6] method, the code available in [50] was run, with some modifications to evaluate the method under the generalized zero-shot setting. The same was made for SAE [8] method, which was re-implemented using Python language, since the original code was written in Matlab. To compare the original results and the reproduced ones, the values followed by an asterisk, in the table 2.3, represent the results that we obtained from the method's code execution. In the same way, the generalized zero-shot learning results are showed in Table 2.4.

The presented results in zero-shot and generalized zero-shot configurations emphasize the effectiveness and superiority of generative methods in almost all datasets. As expected, generative methods perform better since they generate data for the unseen classes, which significantly increases the performance of the method in recognizing novel classes. Regarding the re-implemented methods, we can conclude that the reproduced results are very similar to the values reported in original papers. However, there are slightly different results obtained with the SAE [8] method on AWA1 [23] and AWA2 [2] datasets in standard split (63.6% and 65.9%) compared with the reported results (80.6% and 80.7%), which can be explained by the improvement of visual features made by the authors of SAE [8], as confirmed by Xian et al. [2], which inherently leads to better results in assessing zero-shot learning.

To evaluate performance in the ImageNet [13] dataset and fairly compare the methods that report results for this dataset, distinct dataset class splits were considered, as shown in table 2.5. Thereby, "2H" means that classes are 2-hops away from the original classes concerning the label hierarchy existing in ImageNet [13], which corresponds to 1509 different classes. Analousgly, "3H" means that classes are 3-hops away from the original classes, which makes a total of 7678 different classes. The most challenging split falls in all the remaining approximately 20.000 (All) classes of ImageNet [13] with at least 1 image per-class. As we can notice from the table 2.5, generative-based methods perform better in large-scale scenarios, which clearly indicates that they are a good option to tackle the problem of zero-shot learning. Nevertheless, when the number of classes increases, the performance of all methods tends to reduce.

Table 2.3: Zero-shot learning results on SUN, CUB, AWA1, AWA2, and aPY datasets using standard split (SS) and proposed split (PS).The symbol ("-") means that there are not reported results for that measure/dataset. The values followed by an asterisk are the result of methods re-evaluation task, all others are taken from [2] and from original publications. The results report top-1 accuracy in %.

Zero-Shot Learning										
	SUN		CUB		AWA1		AWA2		aPY	
Method	SS	PS	SS	PS	SS	PS	SS	PS	SS	PS
DAP [23]	38.9	39.9	37.5	40.0	57.1	44.1	58.7	46.1	35.2	33.8
IAP [23]	17.4	19.4	27.1	24.0	48.1	35.9	46.9	35.9	22.4	36.6
CMT [3]	41.9	39.9	37.3	34.6	58.9	39.5	66.3	37.9	26.9	28.0
DeViSE [4]	57.5	56.5	53.2	52.0	72.9	54.2	68.6	59.7	**35.4**	**39.8**
ConSE [27]	44.2	38.8	36.7	34.3	63.6	45.6	67.9	44.5	25.9	26.9
ALE [5]	59.1	58.1	53.2	54.9	78.6	59.9	80.3	62.5	30.9	39.7
ESZSL [6]	57.3	54.5	55.1	53.9	74.7	58.2	75.6	58.6	34.4	38.3
ESZSL* [6]	53.9*	52.3*	54.6*	51.3*	76.3*	56.2*	75.2*	54.5*	34.0*	38.5*
LatEm [7]	56.9	55.3	49.4	49.3	74.8	55.1	68.7	55.8	34.5	35.2
SAE [8]	42.4	40.3	33.4	33.3	**80.6**	53.0	**80.7**	54.1	8.3	8.3
SAE* [8]	47.8*	43.1*	39.2*	38.9*	63.6*	46.7*	65.9*	45.6*	30.1*	28.8*
f-CLSWGAN [1]	–	60.8	–	57.3	–	68.2	–	–	–	–
CVAE-ZSL [9]	–	61.7	–	52.1	–	**71.4**	–	65.8	–	–
f-VAEGAN-D2 [10]	–	65.6	–	**72.9**	–	–	–	**71.1**	–	–
SGMA [35]	**70.5**	**71.0**	**83.5**	68.8	–	–	–	–	–	–

Table 2.4: Generalized zero-shot learning on all reviewed methods for the SUN, CUB, AWA1, AWA2, and aPY datasets. Measure ts = Top-1 Accuracy on Y_u, tr = Top-1 Accuracy on Y_s, H = Harmonic Mean. The symbol ("-") means that there are not reported results for that measure/dataset. The values followed by an asterisk are the result of methods re-evaluation task, all others are taken from [2] and from original publications. The results report top-1 accuracy in %.

Generalized Zero-Shot Learning															
	SUN			CUB			AWA1			AWA2			aPY		
Method	ts	tr	H	ts	tr	H	ts	tr	H	ts	tr	H	ts	tr	H
DAP [23]	4.2	25.1	7.2	1.7	67.9	3.3	0.0	88.7	0.0	0.0	84.7	0.0	4.8	78.3	9.0
IAP [23]	1.0	37.8	1.8	0.2	72.8	0.4	2.1	78.2	4.1	0.9	87.6	1.8	5.7	65.6	10.4
CMT [3]	8.7	28.0	13.3	4.7	60.1	8.7	8.4	86.9	15.3	8.7	89.0	15.9	**10.9**	74.2	**19.0**
DeViSE [4]	16.9	27.4	20.9	23.8	53.0	32.8	13.4	68.7	22.4	17.1	74.7	27.8	4.9	76.9	9.2
ConSE [27]	6.8	**39.9**	11.6	1.6	72.2	3.1	0.4	88.6	0.8	0.5	90.6	1.0	0.0	**91.2**	0.0
ALE [5]	21.8	33.1	26.3	23.7	62.8	34.4	16.8	76.1	27.5	14.0	81.8	23.9	4.6	73.7	8.7
ESZSL [6]	11.0	27.9	15.8	12.6	63.8	21.0	6.6	75.6	12.1	5.9	77.8	11.0	2.4	70.1	4.6
ESZSL* [6]	13.1*	27.3*	17.7*	11.8*	63.0*	20.0*	16.5*	**91.1***	27.9*	6.5*	92.8*	12.2*	2.3*	79.7*	4.6*
LatEm [7]	14.7	28.8	19.5	15.2	57.3	24.0	7.3	71.7	13.3	11.5	77.3	20.0	0.1	73.0	0.2
SAE [8]	8.8	18.0	11.8	7.8	54.0	13.6	1.8	77.1	3.5	1.1	82.2	2.2	0.4	80.9	0.9
SAE* [8]	12.3*	23.1*	16.1*	11.3*	52.6*	18.5*	1.6*	90.7*	3.2*	1.2*	**93.3***	2.4*	0.2*	86.0*	0.3*
f-CLSWGAN [1]	42.6	36.6	39.4	43.7	57.7	49.7	**57.9**	61.4	**59.6**	–	–	–	–	–	–
CVAE-ZSL [9]	–	–	26.7	–	–	34.5	–	–	47.2	–	–	51.2	–	–	–
f-VAEGAN-D2 [10]	**50.1**	37.8	**43.1**	**63.2**	**75.6**	**68.9**	–	–	–	**57.1**	76.1	**65.2**	–	–	–
SGMA [35]	–	–	–	36.7	71.3	48.5	37.6	87.1	52.5	–	–	–	–	–	–

2.7 Conclusions

This chapter provided an overview of the zero-shot learning taxonomy, where a detailed description of the crucial components of the zero-shot problem was focused on. The review of the methods from a temporal perspective, i.e., starting with the oldest and ending with the more recent ones, allowed us to establish a timeline that gives us the perception of how the adopted approaches have evolved. More recently, the generative approaches have proven to be effective in addressing the zero-shot problem, as demonstrated in sec-

Table 2.5: Zero-shot learning results on ImageNet dataset [13]. 2/3 H = Classes with 2/3 hops away from the original classes of ImageNet [13]. All = The remanining 20K classes of ImageNet. The symbol ("-") means that there are not reported results for that measure/dataset.

	Zero-Shot Learning			Generalized Zero-Shot Learning		
Method	2H	3H	All	2H	3H	All
DeViSE [4]	5.25	1.29	0.49	–	–	–
ConSE [27]	7.63	2.18	0.95	–	–	–
CMT [3]	2.88	0.67	0.29	–	–	–
LatEm [7]	5.45	1.32	0.50	–	–	–
ALE [5]	5.38	1.32	0.50	–	–	–
ESZSL [6]	6.35	1.51	0.62	–	–	–
SAE [8]	4.89	1.26	0.56	–	–	–
f-CLSWGAN [1]	11	2.4	1	4.2	1.3	0.5
f-VAEGAN-D2 [10]	**13**	**3.2**	**1.8**	**5**	**1.9**	**0.9**

tion 2.6. In addition, the most widely used datasets for zero-shot learning were also presented and characterized by the number of classes, number of instances, and attributes. Moreover, the importance of choosing a dataset split that does not violate the zero-shot learning assumption was underlined. Finally, a concise benchmark was drawn for each reviewed method on the described datasets, supporting the thesis of the generative approache's superiority on both evaluating settings: ZSL and GZSL.

Chapter 3

Proposed Method

This chapter describes the proposed method for developing the zero-shot learning-based approach to recognize unknown objects using only its textual description. Section 3.1 presents the work plan overview and the description of the tasks to achieve the objectives stated in section 1.2. In section 3.2, we provide the details of the developed evaluation framework for assessing ZSL methods in terms of computational performance. Finally, in section 3.3 we describe our proposed Semantic-Guided Attention Model that aims to learn more discriminative visual features conditioned on the most dominant semantic attributes regarding the input image.

3.1 Work Plan

In order to meet the main objectives of this work, a list of several tasks is proposed. An approximate scheduling for the execution of these tasks is included in table 3.1. The execution of a given task in a given month is marked with a cross (x).

Table 3.1: Work plan scheduling.

Task \ Month	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun
T1 - State of the Art on ZSL	x	x	x						
T2 - Implementation of ZSL Methods			x	x	x	x			
T3 - Evaluation of ZSL Methods					x	x	x		
T4 - Development of a ZSL Method for Low-power Devices						x	x		
T5 - Dissertation Writting			x	x	x	x	x	x	x

Following, we provide a detailed description of each task included in table 3.1.

T1 - State of The Art on ZSL. The initial task consists of the revision of the zero-shot learning literature and taxonomy. This study comprehends the analysis of the accuracy and the computational complexity of the reviewed methods and also the knowledge about the datasets used to evaluate the zero-shot strategies. Furthermore, the major challenges in the ZSL and the evaluation protocols and its metrics are covered in this task.

T2 - Implementation of ZSL Methods. After having the knowledge about the reviewed literature of zero-shot learning, which should give the insights about the most suitable and promising methods for the concerned problem, the implementation of ZSL methods is followed. In particular, the accuracy and the computational complexity of the methods will be taken into account.

T3 - Evaluation of ZSL Methods. The evaluation process of ZSL methods must follow some guidelines to avoid misleading results that can induce a poor evaluation of the models' performance. For this purpose, a study of the evaluation protocol of ZSL approaches is necessary, as stated in task T1. Then, the evaluation of the ZSL methods must be performed on the reviewed datasets to provide the accuracy and time execution metrics under the ZSL and GZSL settings.

T4 - Development of a ZSL Method for Low-power Devices. The ZSL methods' development phase for low-power devices is the union of all acquired knowledge and experiments performed until this phase. The developed strategy should combine all reviewed methods' strengths to overcome the major challenges in zero-shot classification, described in section 2.3. Moreover, the ZSL method must be optimized to run on low-power devices.

3.2 Evaluation of ZSL Methods

In chapter 2, an extensive review of the state-of-the-art in ZSL was performed so that we can identify the crucial points of the ZSL. This way, it was possible to became aware of the main challenges in the ZSL scenarios that can affect the model's performance. On the other hand, we became familiar with the recent advances made towards improving the performance of the models on the most challenging scenarios (GZSL setting) and the efforts that have been made to mitigate the inherent problems that the ZSL methods are faced.

Despite the vast benchmarking that has been done around each novel ZSL method, few works have measured the computational performance of the developed strategy regarding inference time. In addition, we argue that the choice of the CNN-based architecture to extract the visual features may have an impact on the model's performance in terms of both accuracy and image processing speed. Although a vast majority of ZSL works adopt only the *ResNet101* architecture as the feature extractor network for benchmarking purposes, we are interested in evaluating the impact of using lightweight CNN architectures in the ZSL accuracy. Consequently, we decided to develop an open-source evaluation framework for analyzing the accuracy/speed trade-off in the problem of ZSL, allowing us to perceive how the ZSL methods perform in real-world scenarios, specifically when run in low-power devices.

In the next sections, the details of the evaluation framework with regard to its functionality and also the available options are described.

3.2.1 Methods

A set of six state-of-the-art ZSL methods is available to be tested in the developed framework, including ESZSL [6], SAE [8], DEM [39], f-CLSWGAN [1], TF-VAEGAN [51], and

CE-GSZL [52]. The first four methods were covered in chapter 2, however the last two methods, TF-VAEGAN [51] and CE-GSZL [52], are two recent strategies that have achieved impressive results particularly on the most challenging scenario (GZSL). TF-VAEGAN [51] combines the strengths of VAEs and GANs and introduces a feedback module that utilizes a semantic embedding decoder (SED) to improve the generated features during all stages of ZSL framework: training, feature synthesis, and classification. CE-GZSL [52] is a hybrid model that integrates a feature generation model with a Contrastive Embedding (CE) model, and maps both real and synthesized features into a new embedding space, where the GZSL classification is performed. In particular, a Contrastive Embedding (CE) is proposed to exploit the class-wise supervision and also the instance-wise supervision for GZSL.
Regarding the projection methods (ESZSL, SAE, and DEM), we implemented them by ourselves. However, for the generative methods (f-CLSWGAN, TF-VAEGAN, and CE-GZSL) we used the original implementation code available in the *GitHub* repository [53, 54, 55]. Moreover, we made the necessary changes to adapt the code to the datasets' requirements and provide the output results in a specific format.

3.2.2 Datasets

Among the most widely used datasets for zero-shot learning, we considered the *Animals with Attributes 1* (AWA1) [23], *Animals with Attributes 2* (AWA2) [2], *Caltech-UCSD-Birds* (CUB-200-2011) [40], *Attribute Pascal and Yahoo* (APY) [18], *SUN Attributes* (SUN) SUN [46] ,and the most recent *Large-scale Attribute Dataset* (LAD) [56] dataset for our experiments.
For each of these datasets, we provide a set of files with a standardized data structure necessary to run the different ZSL methods. Specifically, we followed the same file pattern of the data provided by [2], which is based on a simple dictionary of objects. Concretely, there are two dictionaries (provided in a Matlab format) to be considered: (1) *att_splits.mat* - a dictionary for storing the class attribute vectors and the instances indexes of the splits for training, validation, and test purposes; and (2) *ResNet101.mat* - a dictionary to store the visual features vectors extracted from a specific CNN architecture and the correspondent class labels. However, if the visual features were extracted from the *MobileNet* architecture, the nomenclature for the dictionary file is *MobileNet.mat*.

3.2.3 Custom Feature Extractor

As stated above, the majority of ZSL works rely on the *ResNet101* architecture for the feature extraction process. However, we want to perceive and analyze the impact of using lightweight networks, such as *MobileNet* and *MobileNetV2*, in the ZSL method's performance compared to the reported results on the *ResNet101* architecture.
Thereby, a set of CNN architectures, including *MobileNet*, *MobileNetV2*, *ResNet101*, *Xception*, and *EfficientNEtB7* were considered to provide a robust analysis of the performance of the ZSL methods in different datasets using state-of-the-art methods, including the

most recent ones.

Although we provide the features for the five above-described CNN architectures, there is an option in our framework for extracting CNN features using one of the available architectures so that can be evaluated in the desired ZSL method.

3.2.4 Optimizing Models using TensorRT

After developing a deep learning model, the deployment phase regards the integration of the developed strategy in any platform or hardware device. However, we should be concerned about the computational performance of the method when the evaluation is performed in a low-power device. Due to the inherent limitations in terms of processing speed in the low-power devices, some techniques should be used to optimize the model towards high performance in the inference stage. One of these techniques is called TensorRT optimization [11]. In a nutshell, TensorRT [11] is a tool designed by Nvidia Corporation for optimizing deep learning models in order to run quickly and efficiently on a GPU, achieving high performance on the inference phase. Moreover, TensorRT works with the most popular deep learning frameworks, such as Tensorflow, PyTorch, and MXNet. To optimize the models, TensorRT performs several transformations and matrix optimizations around the model graph. In the case of TensorFlow is adopted as the development framework, TensorRT operates in the named TF-TRT mode. For example, layers with unused outputs are removed, multiple layers are fused into a single layer (when possible), and normalization and conversion to optimized matrix math are performed according to the specified precision mode: FP32, FP16, or INT8. The precisions lower than FP32 improve the performance of the inference phase.

As depicted in figure 3.1, the top image shows the typical workflow when TensorRT is adopted to optimize a DNN model. The bottom image shows examples of real-world applications when TensorRT is used, including Robots (using a Jetson Nano), Autonomous Vehicles, and large-scale applications.

3.2.5 Computational Performance

Since our primary purpose is to provide a concise benchmarking regarding ZSL method's accuracy and processing speed using visual features obtained through distinct architectures, the measure of elapsed time in both the feature extraction process and inference phase is paramount.

Inference Time in Image Preprocessing Considering that the visual feature extraction process is the first phase in the ZSL pipeline, we are interested in analyzing the time spent in this operation. Moreover, this task is performed using CNN-based networks, which means that the processing speed is directly related to the complexity of the model. In general, the more complex the model the larger the time spent for computing the fea-

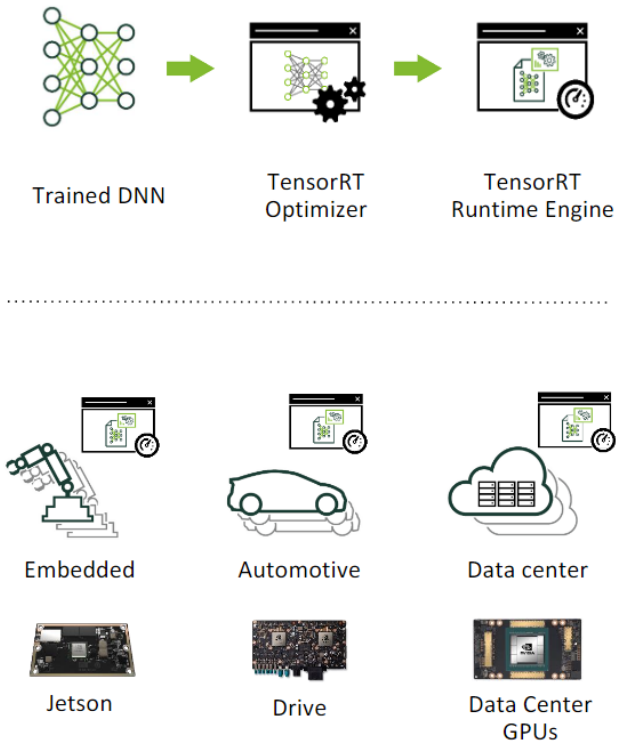


Figure 3.1: The typical workflow of the TensorRT integration (above) and applications in real-world production environments (below) [11].

ture vector. Furthermore, we assume that the preprocessing phase includes all the steps from when the image is acquired until it is transformed into the feature vector format.

The inference time regarding the image preprocessing is computed by the average time that the CNN-based model takes to extract the visual feature for a given input image over N iterations. In order to perceive the variation of the results, the standard deviation was also measured in our essays.

Inference Time in Class Prediction In a similar manner, we evaluate the time spent in the class prediction task considering distinct feature dimensions, ranging from 512 to 4,032. The inference time is calculated as the time spent to predict the class of a single test sample. We averaged the results after N iterations and also computed the standard deviation.

3.2.6 Statistics

For all evaluated ZSL algorithms, we provide the results on both ZSL and GZSL settings. For the standard ZSL setting, we measure the Top-1 accuracy, whereas in the generalized setting we compute the Harmonic mean, following the proposed evaluation protocol in [2].

3.2.7 Usability

One of the goals of the proposed evaluation framework is to provide an easy way to evaluate the various ZSL algorithms. Consequently, we adapted each of the ZSL methods so that it is possible to execute them from a single line of code. It will only be necessary to invoke the desired method, either with or without configuration parameters, since all methods have a set of parameters assigned by default. Following, there is an example of how to run SAE algorithm using the visual features extracted from *MobileNetV2* architecture:

```
from utils.engine import SAE

SAE(dataset="CUB", filename="MobileNetV2")

Output:

Evaluating on CUB...
Mode: V2S
[ZSL] Top-1 Accuracy (%): 34.73 %
[GZSL] Accuracy (%) - Seen: 43.77 %, Unseen: 9.21 %, Harmonic: 15.22 %
```

3.3 Proposed Semantic-Guided Attention Model

Generative approaches have proven to be effective in addressing the zero-shot learning problem [10, 39, 52]. Furthermore, it has been discovered that the learning of discriminative visual features, especially in fine-grained scenarios, improves significantly the performance of zero-shot classification [35].

The work done by Zhu et al. [35] brings a new approach for learning discriminative visual features guided by the semantic attributes annotated per image. They propose a multi-attention localization model for producing attention maps that capture the discriminative parts of the image regarding the semantic attributes. Then, these crucial regions are cropped and fed into a CNN-based network to extract the visual feature vector for each discriminative part. However, cropping such parts requires an additional network for learning the approximate region to perform the cropping operation. Inspired by the work done by Liu et al. [57], we argue that the Grad-CAM [58] technique can be exploited for producing class-discriminative localization maps that assembles the regions where a specific class attribute is present. Therefore, we can obtain an Attribute Attention Map (AAM) through the maximum operation over each produced Grad-CAM map. Then, the new semantic-guided feature map is obtained with both global category-level and the local attribute-level discrimination feature map.

In order to combine the strengths of generative methods with the idea of producing more class-discriminative visual features using the Grad-CAM technique, we propose the fusion of a Semantic-Guided Attention Model (SGAM) with a state-of-the-art generative approach. In addition, the visual features produced by the semantic-guided attention model will enable the generation of more accurate and class-discriminative visual features

for the unseen classes using the generative model. Consequently, the major challenges in ZSL that were discussed in section 2.3 can be notably alleviated, namely domain-shift, bias, and hubness.

The proposed method is depicted in figure 3.2, and it consists essentially of three main phases: (1) Attribute Prediction, (2) Attribute Attention Map Generation, and (3) Training of a generative ZSL approach using semantic-guided features. In the following sections, we will discuss the details behind each phase.

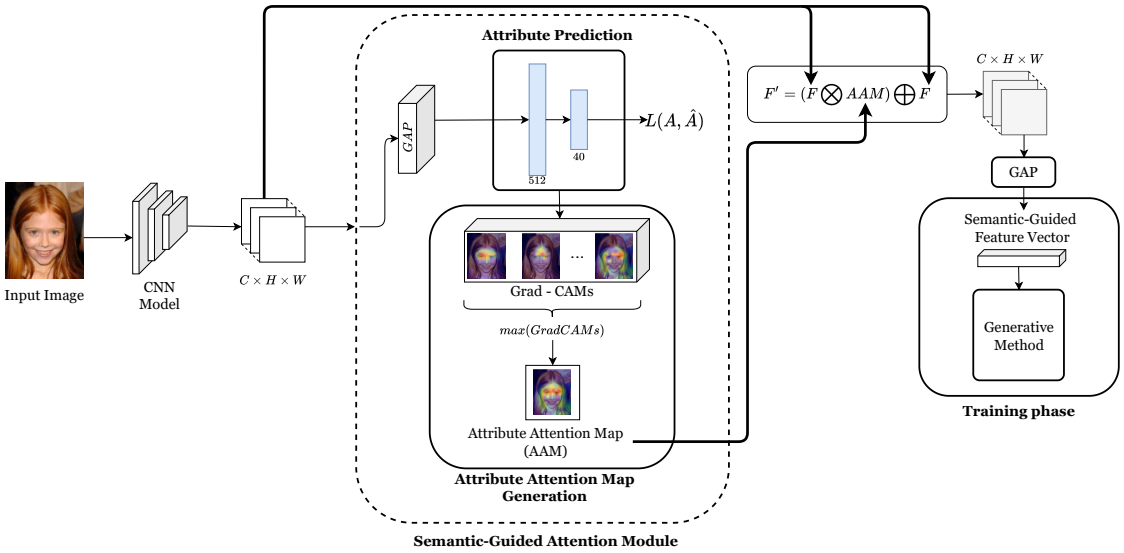


Figure 3.2: **Proposed Semantic-Guided Attention Model**. The input image is fed into a CNN architecture with a custom top layers model that acts as an attribute classifier. After predicting the N attributes for the input image, the Grad-CAM maps are generated regarding the N most influential attributes. The Attribute Attention Map (AAM) is computed as the maximum operation over the N Grad-CAM maps. Finally, a new feature maps is calculated by weighting the original feature maps over the produced AAM. The final semantic-guided feature vector is the result of the average pooling operation over the new feature maps. The generative method is trained using the semantic-guided feature vectors. Hence, the trained generator is used to synthesize the class-discriminative samples for the unseen classes. Finally, a softmax classifier is trained with the instances of both seen and unseen classes to infer the final predictions.

3.3.1 Attribute Prediction

The first stage of our proposed method consists of learning an attribute prediction model. The idea is to obtain the confidence score for each attribute given an input image, so that the final convolutional layers of the network can learn to identify the important parts of the image related to the predicted attributes [58].

We adopted the VGGFace model as the backbone, however we reconstructed the model top layers to match the size of the attribute prediction model layers. The first layer has the size of the channels of the last convolutional layer, followed by a sigmoid activation layer with N neurons, where N corresponds to the number of the attributes to be predicted.

As depicted in figure 3.2, the input image is fed into the CNN-based network, and the result is a feature map with the size of $C \times H \times W$. The feature map is down-sampled to $C \times 1 \times 1$ by average-pooling operation and reshaped into $C \times 1$ to match the dimension of the first layer of the attribute prediction model. The output vector is then normalized by

a sigmoid activation layer, and it gives the probability confidence score of each attribute. The model was optimized by the Adadelta algorithm with the default hyperparameters. Furthermore, we performed data-augmentation on both training and validation set. The chosen loss function was the binary cross-entropy loss, which is defined as:

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log(1 - \hat{y}_i) \quad (3.1)$$

where N is the output size, $\hat{y}_i$ is the predicted value, and y_i is the corresponding target value.

3.3.2 Gradient-weighted Class Activation Mapping

Gradient-weighted Class Activation Mapping (Grad-CAM) [58] is a technique that allows generating a class-discriminative localization map using any CNN-based network. Through the use of the gradient information of any target class, Grad-CAM produces a kind of visual explanation regarding the decisions taken by the network in predicting a specific target class. This technique is a generalization of the Class Activation Mapping (CAM) [59] approach. The main difference between Grad-CAM and CAM is that the CAM requires a particular kind of CNN architecture. Specifically, an architecture that performs global average pooling (GAP) over convolutional maps, followed by the prediction layer (i.e., conv feature maps $\rightarrow$ GAP $\rightarrow$ softmax layer) [58]. In contrast, Grad-CAM can be applied to a wide range of CNN networks without requiring modifications in the network architecture.
The pipeline for producing a class-discriminative localization map Grad-CAM for any image embraces three main steps, which are summarized below.

1. **Calculate the gradient**. The gradient is calculated for the score y^c of the class c with respect to feature maps A^k of the last convolutional layer. The gradient matrix G for the class c is defined as follows:

$$G_k = \frac{\partial y^c}{\partial A^k} \quad (3.2)$$

   with $G_k \in \mathbb{R}^{u \times v}$, $A^k \in \mathbb{R}^{u \times v}$, and $k \in \mathbb{N}$.

2. **Calculate the neuron importance weights** α_k^c. The gradients are then average-pooled in order to obtain the neuron importance weights α_k^c that captures the "importance" of the feature map k regarding the class c.

$$\alpha_k^c = \frac{1}{Z}\sum_i \sum_j \frac{\partial y^c}{\partial A^k} \quad (3.3)$$

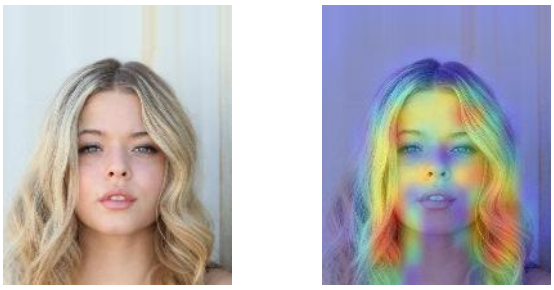
and $\alpha_k^c \in \mathbb{R}^{1\times 1}$.

3. **Calculate the final Grad-CAM map**. The computed weights α_k^c are then weighted with each of the feature maps, followed by a (Rectified Linear Unit) ReLU operation to obtain the final Grad-CAM map:

$$GradCAM_c = ReLU\left(\sum_k \alpha_k^c A^k\right) \quad (3.4)$$

and $GradCAM_c \in \mathbb{R}^{u\times v}$.

The ReLU is applied to the linear combination due to the fact that final Grad-CAM map only captures the features that "have a positive influence on the class of interest" [58]. Moreover, the final Grad-CAM map can be viewed as a heat-map that highlights the regions where the target class is present. Figure 3.3 shows the generated Grad-CAM map for an image taken from CelebA [12] dataset.



Original Image Grad-CAM "Wavy Hair"

Figure 3.3: Original image (at left) representing a woman's face. Modified image (at right) overlapped with the generated Grad-CAM highlighting the face attribute "Wavy Hair". The highlighted regions on the heat-map evidence that the network "focused" on the "hair" region to predict the "Wavy Hair" class.

However, the process of generating Grad-CAMs for each of the attributes can be computationally expensive. A suitable way to obtain good results is to select only the top-D most influential attributes [57]. In this way, it is guaranteed that the final attribute attention map (AAM) contains the sensitive information of the top-D most discriminative attributes. Figure 3.4 shows the generated Grad-CAMs for the top-10 most influential attributes regarding the input image. It is interesting to observe that the final generated AAM is an almost perfect outline of the face image due to the maximum operation performed along the produced Grad-CAM maps.

3.3.3 Calculate Discriminative Features

The Attribute Attention Map (AAM) is used jointly with the feature maps of the last convolutional layer to compute a new feature map to obtain the final semantic-guided feature vector.

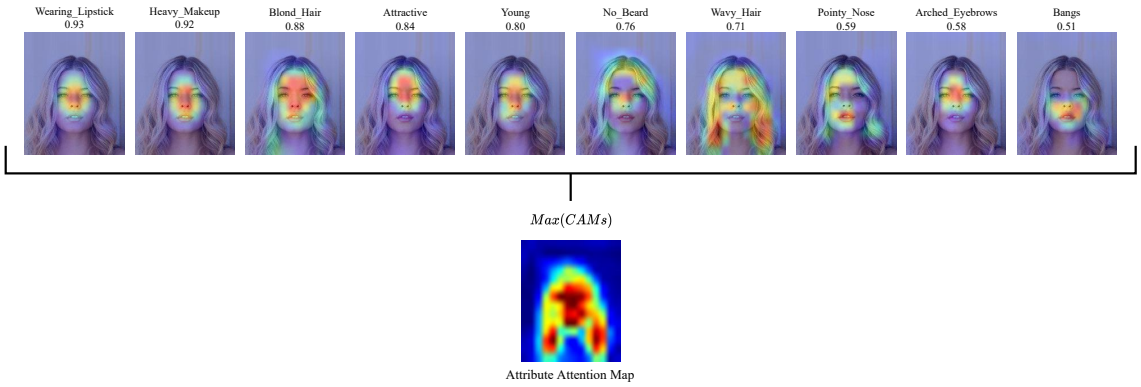


Figure 3.4: **Pipeline for generating the Attribute Attention Map (AAM) for an example face image of the CelebA [12] dataset**. Each of the generated Grad-CAM focuses on the face region that is visually closest to the attribute prediction. The final AAM captures all the Grad-CAM highlighted regions, resulting in an almost perfect face outline.

Actually, the new feature maps $F' = (F \bigotimes AAM) \bigoplus F$ are an improvement of the original feature maps F, since discriminative information is added by the AAM. Thus, the semantic-guided feature vector f is calculated by the global average pooling operation to F' in order to match the dimension of the input layer of the generative method ($C \times 1$). More formally:

$$f = GAP\left(\left(F \bigotimes AAM\right) \bigoplus F\right) \quad (3.5)$$

where GAP stands for Global Average Pooling operation.

3.3.4 Generative Method

After computing the discriminative visual features, these can be used as input for a ZSL model. As discussed before, generative methods are currently the state-of-the-art in ZSL due to their effectiveness particularly in the generalized scenario. For this reason, our model is flexible to incorporate any generative method. However, a non-generative method can also be used. Despite the original purpose of the GAN networks is related with the process of generating images, in ZSL problems, the generative approaches aim to synthesize visual feature vectors conditioned on the given attributes. Furthermore, Xian et al. [1] proved that generating CNN features instead of images leads to a significant increase in performance of both ZSL and GZSL settings since visual features retain more discriminative information than a synthesized image produced by a GAN.
Finally, the synthesized feature vectors are then assigned to each unseen class to train a softmax classifier jointly with the samples of seen classes to infer the final predictions.

Chapter 4

Results and Discussion

This chapter presents an extensive benchmark regarding the computational performance of the ZSL approaches in low-power devices. The experiments results under the ZSL and GZSL settings on the testbed datasets using the proposed evaluation framework are discussed along the section 4.2. Furthermore, the results of the proposed Semantic-Guided Attention strategy are also discussed in section 4.3.

4.1 Materials

4.1.1 Hardware

Experiments were performed in a Desktop Computer without GPU, and two small low-power devices, namely a Raspberry Pi 4 Model B and a Jetson Nano Developer Kit. The hardware specifications are given in table 4.1.

Table 4.1: Hardware Specification.

	Desktop	**Raspberry Pi 4B**	**Jetson Nano Dev Kit**
CPU	Intel® Core™ i7-10700U CPU @ 2.90GHz × 16	Broadcom BCM2711 Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz	Quad-core ARM A57 @ 1.43 GHz
GPU	Intel UHD Graphics 630	-	128-core Maxwell
RAM	32GB	4GB	4 GB
Storage	1TB SSD	64GB microSD	64GB microSD
OS	Pop_OS! 20.10 64-bit	Raspbian	Ubuntu 18.04.5 LTS

4.1.2 Methods

A set of six state-of-the-art ZSL methods, including ESZSL [6], SAE [8], DEM [39], f-CLSWGAN [1], TF-VAEGAN [51], and CE-GSZL [52] were selected to our essays, as previously mentioned in section 3.2.1.

4.1.3 Datasets

As described in section 3.2.2, six benchmark datasets for ZSL were used (AWA1 [23], AWA2 [2], CUB [40], APY [18], SUN [46], and LAD [56]). Among the selected datasets, LAD [56] is the most recent dataset, comprising $78,017$ images and 359-dimensional attribute annotations. Details of dataset statistics are present in Table 4.2.
Visual features are extracted for all datasets with *ResNet101*, *MobileNet*, *MobileNetV2*, *Xception*, and *EfficientNetB7* pre-trained on ImageNet-1K without fine-tuning. However, it is not possible for AWA1 dataset due to the unavailability of the original images.

Table 4.2: Statistics for datasets.

Dataset	No. Classes	No. Instances	No. Attributes
CUB-200-2011 [40]	200	11,788	312
SUN Attibutes [46]	717	14,340	102
AWA1 [23]	50	30,475	85
AWA2 [2]	50	37,322	85
aPascal-aYahoo [18]	32	15,339	64
LAD [56]	230	78,017	359

Moreover, we adopt the Proposed Split (PS) [2] to ensure that none of the test classes appear in the dataset used to pre-train the DNN model. For semantic embeddings, we adopt the class-level attributes provided by [2] for AWA1 (85-dim), AWA2 (85-dim), SUN (102-dim), CUB (312-dim), and APY (64-dim). In case of the LAD dataset, we use the 359-dimensional binary attribute annotations provided by the authors. We conduct all experiments under the inductive setting, in which only labeled instances of seen classes are considered at the training phase [2].

4.2 Results

4.2.1 Evaluation Protocols

In order to measure the performance of ZSL methods, we follow the unified evaluation protocol [2] to assess the methods in the ZSL and GZSL settings. We adopt the Top-1 accuracy to measure the average per-class top-1 accuracy under the ZSL setting. In the GZSL setting, the average per-class top-1 accuracy is computed on training (y^{tr}) and test (y^{ts}) classes. Thus, harmonic mean is computed for the training and test accuracies as $H = \frac{2 * acc_{y^{tr}} * acc_{y^{ts}}}{acc_{y^{tr}} + acc_{y^{ts}}}$.

4.2.2 ZSL Methods: Visual Feature Extraction Cost

The first step in a typical computer vision approach is the description of an image using compact representation. To this end, a common approach uses a Convolutional Neural Network (CNN) to extract the visual information of the image. A feature vector is extracted from one of the last convolutional layers, followed by a pooling operation. Among the vast panoply of the CNN-based networks, a set of distinct CNNs, including lightweight networks, was evaluated in terms of computational cost regarding the time consumed in the feature extraction process using different hardware devices. The results are reported in table 4.3 and show that depending on the architecture chosen, the execution time varies significantly. However, it is possible to perform feature extraction on Raspberry Pi 4B in less than 400 ms if either *MobileNet* or *MobileNetV2* architecture is chosen. The widely used architecture in the ZSL problem, the *ResNet101*, takes more $1,200$ ms than lightweight networks, which is a considerable difference. As expected, the scenario reverses when Jetson Nano is used. Due to its GPU add-on, it is possible to perform feature extraction in less than 40 ms using *MobileNet* and *MobileNetV2* architectures. Moreover, it is worth mentioning that the CNN models were optimized using TensorRT in the case

of Jetson Nano in order to effectively run it on GPU, which significantly boosts the speed on inference time, as mentioned in chapter 3. Nevertheless, even some TensorRT optimized models fail to run in Jetson Nano for a particular type of CNN architectures due to memory issues.

Table 4.3: **Elapsed time in the visual feature extraction process on Desktop, Raspberry Pi 4B (R-PI 4B) and Jetson Nano devices**. Execution time is presented in milliseconds (ms) with the format AVG ± STD. The feature dimension and the size occupied in the disk by the model is also reported.

	Execution Time				
Architecture	Desktop	R-PI 4B	Jetson Nano	Features Dimension	Size (MB)
MobileNet	**25.57**±3.17	310.52±9.80	**29.58**±3.14	1024	16
MobileNetV2	27.59±3.38	**297.63**±8.54	33.04±20.11	1280	14
InceptionV3	33.80±2.81	609.23±3.54	143.28±29.19	2048	92
ResNet50V2	38.07±3.13	887.86±5.27	160.87±1.78	2048	98
NASNetMobile	39.67±2.35	370.10±5.79	111.81±20.40	1056	23
ResNet50	40.25±3.15	968.05±17.31	164.69±3.28	2048	98
Xception	43.43±3.29	1081.18±11.07	155.75±23.21	2048	88
ResNet101V2	54.13±3.09	1655.37±15.72	-	2048	171
DenseNet201	54.79±2.86	1404.16±39.72	-	1920	80
ResNet101	57.46±2.99	1639.46±76.71	-	2048	171
VGG16	59.73±4.10	2046.51±15.03	221.75±16.96	512	528
VGG19	69.15±2.42	2557.60±6.12	-	512	549
EfficientNetB7	86.54±0.85	2436.62±106.28	-	2560	256
NASNetLarge	95.67±2.98	2115.31±16.84	-	4032	343

In conclusion, the lightweight architectures have superior performance using low-power devices to perform the feature extraction. However, it is essential to assess how this particular type of network impacts ZSL accuracy in the testbed datasets.

4.2.3 ZSL Benchmarking

As previously mentioned, most ZSL works adopt the *ResNet101* as the standard architecture for visual feature extraction. Despite that, we report the performance of ZSL approaches when trained with visual features obtained from different CNN architectures, including lightweight architectures, to perceive how the accuracy balances when adopting a different CNN for the feature extraction process.

Accordingly, five different CNN architectures were selected to conduct the experiments, namely *MobileNet*, *MobileNetV2*, *Xception*, *ResNet101*, and *EfficientNetB7*. The *MobileNet* and *MobileNetV2* are two lightweight architectures; the *Xception* is the most lightweight architecture with a feature dimension of 2,048; the *ResNet101* is the most used network in the ZSL benchmarking, and *EfficientNetB7* is a recent proposed CNN architecture with superior complexity.

The ZSL methods were re-trained using the features extracted from the above-referred networks, when possible. The hyper-parameters of each approach were adjusted in the training phase using the validation data. Finally, each method was evaluated in six datasets commonly used in ZSL (AWA1, AWA2, CUB, APY, SUN, and LAD) under the standard and generalized settings.

AWA1 The results on AWA1 are depicted in Figure 4.1. We carried out the experiments using only the *ResNet101* features provided by [2], due to the unavailability of original

images of AWA1, as mentioned in section 4.1.3. The results show the superiority of the TF-VAEGAN model, which achieves a top-1 accuracy of 71.56% in recognizing unseen classes. In the GZSL setting, the best Harmonic mean is also attained by the TF-VAEGAN model with the value of 65.72%.

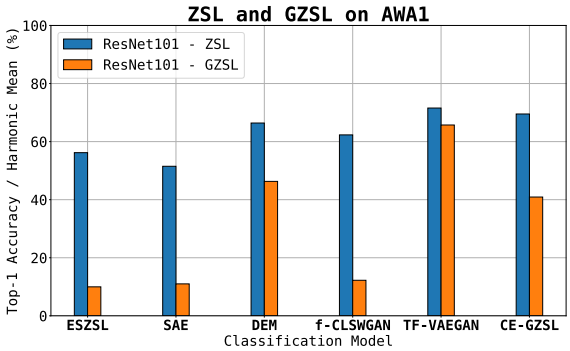
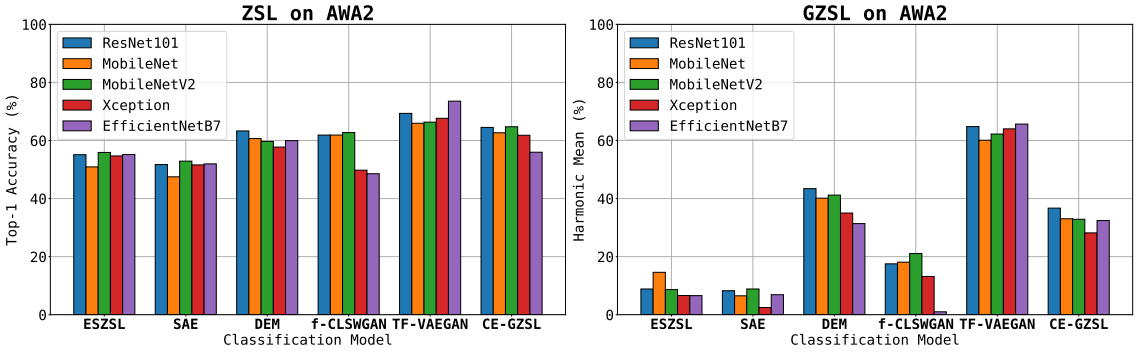


Figure 4.1: ZSL and GZSL performance (%) on AWA1 dataset.

AWA2 Conversely, the AWA2 dataset provides all the original images used to construct the dataset, allowing the extraction of visual features.
According to the results shown in figure 4.2a, in standard setting, it is highlighted the performance of the TF-VAEGAN model, which reaches the best performance (73.55%) when the features extracted from *EfficientNetB7* architecture are considered. However, the features from the *EfficientNetB7* perform worse, on average (56.83% vs 60.97% of the *ResNet101*).
Regarding the generalized setting, figure 4.2b shows that TF-VAEGAN method achieves the best performance (65.66%) by a large margin compared to the other methods. The higher value is also achieved when the *EfficientNetB7* features are considered.



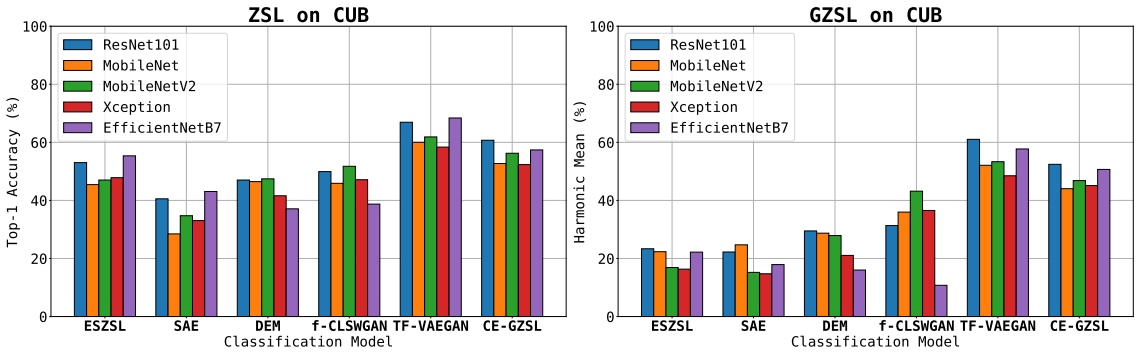
(a) ZSL performance (%) by classification model and CNN architecture on AWA2 dataset.

(b) GZSL performance (%) by classification model and CNN architecture on AWA2 dataset.

Figure 4.2: Accuracy of ZSL approaches on the AWA2 dataset when using different CNN architectures for visual feature extraction.

CUB Figure 4.3a presents the ZSL results for the CUB dataset. Once again, the TF-VAEGAN method attains the best results (66.92% considering the *ResNet101* features, 61.87% with *MobileNetV2* features, and 68.39% using the *EfficientNetB7* features). There

is a significant discrepancy between our results and the reported results in [52] for the CUB dataset. Nevertheless, this gap can be explained by the fact that authors in [52] adopt 1,024-dimensional class embeddings generated from textual descriptions as the semantic descriptors for the CUB dataset instead of working with the original 312-dimensional semantic attributes provided by [2].
Concerning the GZSL results on the CUB dataset, the two most recent generative models (TF-VAEGAN and CE-GZSL) take higher performance. TF-VAEGAN model achieves a Harmonic mean of 61.04%. The second-best model, the CE-GZSL, attains a Harmonic mean of 52.43%. These two results are obtained through the use of *ResNet101* features.



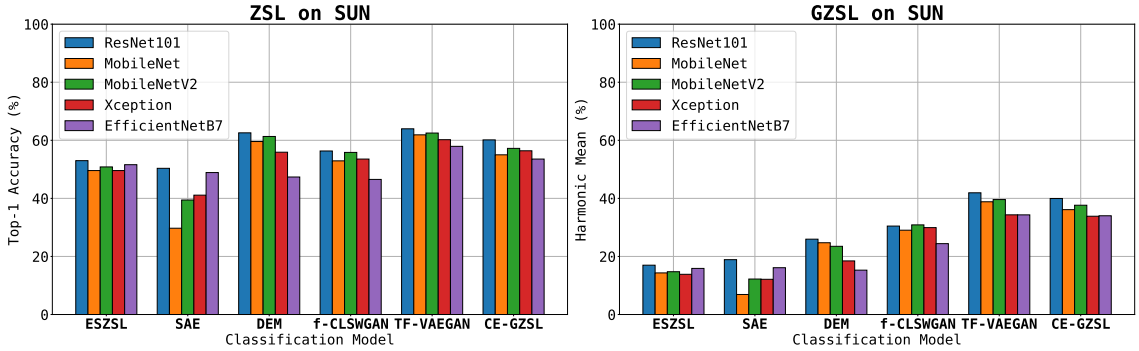
(a) ZSL performance (%) by classification model and CNN architecture on CUB dataset.

(b) GZSL performance (%) by classification model and CNN architecture on CUB dataset.

Figure 4.3: Accuracy of ZSL approaches on the CUB dataset when using different CNN architectures for visual feature extraction.

SUN Figure 4.4a shows the results on the SUN dataset under the standard evaluation setting. The *ResNet101* features have a better performance in all classification models, with a Top-1 accuracy higher than 50% in all cases. The TF-VAEGAN achieves better results (61.29%, on average), followed by the DEM model (57.35%, on average).
About the GZSL results on the SUN dataset, figure 4.4b shows the superiority of the generative models, as expected. Once again, the *ResNet101* features reach the best results in all ZSL methods, except for the f-CLSWGAN model, in which the *MobileNetV2* features achieve high performance.

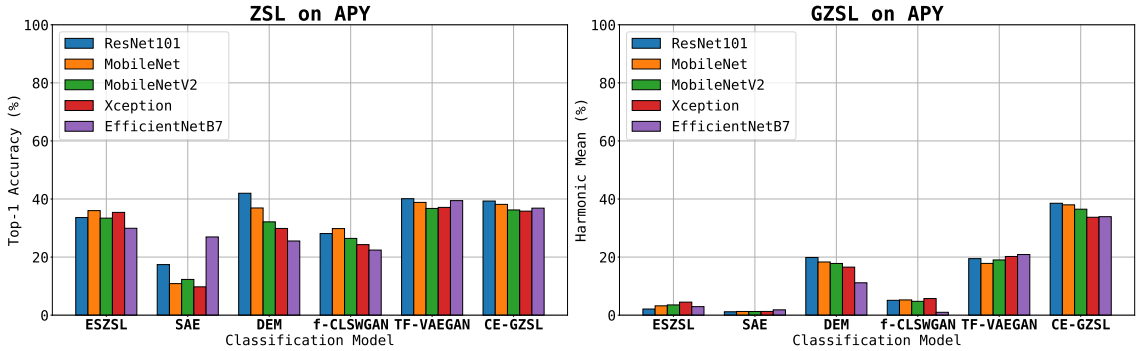
APY According to the authors of the LAD dataset, the APY dataset can be considered unsuitable for validating ZSL methods because it only contains 32 different categories of objects. However, we evaluate the methods on the APY dataset to provide a more concise benchmark. The results presented in figure 4.5a evidence that only the DEM method achieves a top-1 accuracy higher than 40%. Even generative methods perform worse because it is hard to synthesize discriminative features for the unseen classes when there are few training classes, and consequently training samples, to fine-tune the classifiers.
However, in the generalization setting, CE-GZSL model achieves an harmonic mean closer to 40%, surpassing the rest of the models by a large margin.



(a) ZSL performance (%) by classification model and CNN architecture on SUN dataset.

(b) GZSL performance (%) by classification model and CNN architecture on SUN dataset.

Figure 4.4: Accuracy of ZSL approaches on the SUN dataset when using different CNN architectures for visual feature extraction.



(a) ZSL performance (%) by classification model and CNN architecture on APY dataset.

(b) GZSL performance (%) by classification model and CNN architecture on APY dataset.

Figure 4.5: Accuracy of ZSL approaches on the APY dataset when using different CNN architectures for visual feature extraction.

Apart from the graphical view of the results, we provide the numerical results for all the evaluated methods in the table 4.4.

LAD In contrast to the remaining datasets, the performance value of the LAD dataset corresponds to the average accuracy on all super-classes (Aminals, Fruits, Vehicles, Electronics, and Hairstyles) on the given five splits. Since the data provided by the authors of the LAD dataset does not include the appropriate data splits for evaluating the dataset under the GZSL setting, we have created the required data splits to this end. The splits are available at `https://github.com/CristianoPatricio/zsl-methods`, following the same data structure of the splits provided by [2].

In this dataset, we only conduct experiments using visual features extracted from *MobileNetV2* architecture, due to its reduced computational cost, and for surpassing the *MobileNet* regarding the ZSL accuracy.

The results in table 4.5 evidence that for all ZSL methods, both the “Animals” and “Vehicles” classes attain the best accuracy in the standard setting. On the other hand, the ZSL algorithms have more difficulty in recognizing fruits, electronics or hairstyles, for being more fine-grained classes.

Table 4.4: ZSL and GZSL accuracy for all ZSL methods evaluated on the AWA2, CUB, SUN and APY datasets using different CNN architectures for the visual feature extraction.

Methods	Architecture	AWA2				CUB				SUN				APY			
		ZSL	GZSL			ZSL	GZSL			ZSL	GZSL			ZSL	GZSL		
		MCA	U	S	H	MCA	U	S	H	MCA	U	S	H	MCA	U	S	H
ESZSL [6]	ResNet101	55.11	4.66	87.07	8.86	53.02	14.29	63.73	23.35	52.99	12.15	28.22	16.99	33.62	1.07	72.24	2.11
	MobileNet	50.91	8.04	79.32	14.60	45.45	14.22	52.13	22.34	49.58	10.49	22.64	14.33	36.00	1.65	66.23	3.21
	MobileNetV2	55.89	4.56	83.94	8.65	47.02	9.96	55.71	16.89	50.83	10.76	23.37	14.74	33.41	1.81	68.91	3.52
	Xception	54.68	3.44	86.67	6.61	47.83	9.55	56.23	16.33	49.58	9.44	25.93	13.85	35.39	2.32	69.75	4.48
	EfficientNetB7	55.16	3.42	87.41	6.57	55.35	13.45	63.37	22.19	51.60	11.25	26.98	15.88	29.92	1.50	69.62	2.93
SAE [8]	ResNet101	51.71	4.34	85.39	8.26	40.55	14.10	52.55	22.24	50.35	15.97	23.10	18.89	17.41	0.6	18.02	1.16
	MobileNet	47.49	3.40	77.62	6.51	28.47	15.87	55.74	24.70	29.72	5.69	8.76	6.90	10.86	0.69	8.32	1.27
	MobileNetV2	52.89	4.66	87.07	8.86	34.73	9.21	43.77	15.22	39.44	9.93	15.89	12.22	12.3	0.71	5.25	1.25
	Xception	51.59	1.25	87.64	2.47	33.06	8.84	44.32	14.74	41.11	9.65	16.32	12.13	9.75	0.71	6.68	1.28
	EfficientNetB7	51.94	3.59	85.70	6.89	43.07	10.59	57.61	17.90	48.89	12.71	22.05	16.12	26.93	0.93	53.70	1.83
DEM [39]	ResNet101	63.29	29.21	84.60	43.42	47.02	21.56	46.66	29.49	62.57	20.0	36.94	25.95	**41.98**	11.54	71.28	19.86
	MobileNet	60.66	26.59	81.87	40.14	46.48	20.21	49.61	28.72	59.58	18.61	36.78	24.72	36.91	10.59	67.87	18.31
	MobileNetV2	59.68	27.25	84.37	41.19	47.43	20.30	44.59	27.89	61.32	18.26	32.91	23.49	32.13	10.27	66.71	17.80
	Xception	57.73	21.98	86.09	35.02	41.6	15.78	31.66	21.06	55.9	13.89	27.52	18.46	29.86	9.61	59.53	16.54
	EfficientNetB7	59.95	19.14	87.04	31.38	37.1	11.49	26.41	16.02	47.36	12.15	20.58	15.28	25.54	6.37	44.10	11.14
f-CLSWGAN [1]	ResNet101	61.87	9.69	**90.89**	17.52	49.92	20.93	62.45	31.35	56.32	28.47	32.71	30.46	28.1	2.64	76.92	5.11
	MobileNet	61.87	10.07	89.46	18.10	45.89	27.13	53.39	35.98	52.92	33.13	25.85	29.04	29.81	2.71	76.14	5.24
	MobileNetV2	62.73	11.94	89.73	21.07	51.75	38.46	49.22	43.18	55.83	36.81	26.59	30.87	26.43	2.45	**78.61**	4.76
	Xception	49.78	7.11	89.96	13.17	47.13	27.78	53.39	36.54	53.54	29.03	30.89	29.93	24.29	2.99	64.41	5.71
	EfficientNetB7	48.56	0.50	90.12	1.0	38.73	5.96	55.39	10.76	46.53	25.69	23.26	24.41	22.41	0.5	75.82	0.98
TF-VAEGAN [51]	ResNet101	69.34	**57.60**	74.04	64.79	66.92	**57.91**	64.52	**61.04**	**63.96**	46.60	**38.10**	**41.92**	40.1	11.15	76.54	19.46
	MobileNet	65.93	52.61	70.06	60.09	60.02	48.14	56.76	52.10	61.88	45.63	33.80	38.83	38.83	10.24	68.57	17.82
	MobileNetV2	66.32	53.33	74.65	62.22	61.87	50.55	56.43	53.33	62.5	**46.88**	34.26	39.59	36.75	10.99	70.28	19.0
	Xception	67.64	53.90	78.76	64.00	58.37	44.52	53.24	48.49	60.21	39.51	30.35	34.33	37.11	11.81	72.20	20.21
	EfficientNetB7	**73.55**	54.92	81.62	**65.66**	**68.39**	51.43	**65.77**	57.72	57.92	36.04	32.75	34.32	39.44	12.05	78.25	20.88
CE-GZSL [52]	ResNet101	64.5	23.03	90.46	36.71	60.72	56.75	48.72	52.43	60.14	45.14	35.85	39.96	39.3	**31.07**	50.69	**38.53**
	MobileNet	62.66	20.39	86.98	33.04	52.71	39.89	49.17	44.04	55.0	40.14	32.83	36.12	38.15	29.40	53.67	37.99
	MobileNetV2	64.73	20.18	88.56	32.87	56.24	48.37	45.38	46.83	57.22	44.10	32.83	37.64	36.23	30.82	44.72	36.49
	Xception	61.8	16.71	89.80	28.18	52.34	39.22	53.14	45.13	56.39	38.33	30.31	33.85	35.82	11.81	72.20	20.21
	EfficientNetB7	55.96	19.76	90.35	32.43	57.39	50.24	51.14	50.68	53.54	37.15	31.36	34.01	36.85	25.87	49.19	33.91

In general, the best accuracy on both evaluation settings is obtained by the TF-VAEGAN model. However, disregarding the performance of SAE, the remaining accuracy results are well balanced.

Concerning the generalized setting, the generative methods, except f-CLSWGAN, outperforms the remaining ZSL methods significantly, achieving a harmonic mean upper to 40%.

Table 4.5: ZSL and GZSL results on LAD dataset using MobileNetV2 features.

Methods	Animals			Fruits			Vehicles			Electronics			Hairstyles			Average			
	ZSL	GZSL		ZSL	GZSL		ZSL	GZSL		ZSL	GZSL		ZSL	GZSL		ZSL	GZSL		
	MCA	U	S	MCA	U	S	MCA	U	S	MCA	U	S	MCA	U	S	MCA	U	S	H
ESZSL [6]	**65.72**	1.33	75.28	40.03	4.31	65.12	65.90	5.22	74.25	37.80	1.85	73.18	**41.19**	7.02	20.76	50.13	3.95	61.72	7.42
SAE [8]	45.09	4.26	76.85	27.67	6.15	49.06	55.73	5.49	70.78	34.76	5.19	68.66	38.04	9.04	13.51	40.26	6.03	55.77	10.88
DEM [39]	62.10	17.14	74.21	42.12	8.12	53.08	65.50	20.68	65.13	40.53	8.06	68.76	41.71	6.26	15.21	50.39	12.05	55.28	19.79
f-CLSWGAN [1]	58.12	10.19	**84.22**	35.49	2.15	**73.28**	61.77	5.77	**83.56**	33.61	2.94	**80.33**	38.48	0.5	**31.65**	45.49	4.31	70.61	8.12
TF-VAEGAN [51]	64.53	55.90	60.57	**48.07**	**33.89**	57.49	**67.06**	**53.21**	63.80	**40.62**	**26.93**	67.05	40.93	30.68	12.23	52.24	40.12	52.23	45.38
CE-GZSL [52]	64.27	**57.50**	56.3	37.79	27.25	52.57	60.70	45.22	59.2	33.75	25.38	60.31	38.90	**30.79**	10.47	47.08	37.22	47.77	41.84

Finally, and according to the authors of the LAD dataset, the evaluation of the performance of the ZSL algorithms under different data splits, each of them containing different seen/unseen classes, is more reliable due to the distinctive correlations of the classes.

4.2.4 ZSL Methods: Inference Time

This section intends to analyze the time consumed by different ZSL methods for classifying a single test image without including the elapsed time in the feature extraction process. The experiments were conducted in different hardware devices using six ZSL methods, and considering visual features of variable dimensions. The results are reported in table 4.6, and we can deduce that, in general, ZSL approaches are extremely fast when the feature extraction phase is omitted. It can be observed that the elapsed time of class prediction of the attribute-based methods (DAP and IAP) and also the projection-based methods (SAE, ESZSL) vary slightly. However, in DEM, the consumed time for predicting the class

label appears to be increasing proportionally with the dimension of the visual feature. Actually, in DEM, the visual features space is adopted as the embedding space, where the class prediction is performed, and in general, the visual features space has a larger dimension than the semantic space, indicating that the consumed time in class prediction task increases with the visual feature dimension.
On the other hand, a typical approach for class prediction in generative approaches (f-CLSWGAN) is the use of a softmax classifier, which can be extremely fast when optimized to run on GPU. The results show that Jetson Nano has a very competitive performance compared to the results on Desktop since the evaluated models were optimized using TensorRT. As expected, Jetson Nano excels the performance of the Raspberry Pi 4B, in general.

Table 4.6: **Elapsed time in class prediction considering several visual features dimensions**. In general, ZSL approaches are extremely fast when the feature extraction phase is omitted. As expected, Jetson Nano has a greater performance when compared to Raspberry PI 4B. The results are reported in milliseconds with the format: AVG $\pm$ STD.

	Visual Features Dimension											
	512			**1024**			**2048**			**4032**		
Method	Desktop	R-PI 4B	Jetson Nano	Desktop	R-PI 4B	Jetson Nano	Desktop	R-PI 4B	Jetson Nano	Desktop	R-PI 4B	Jetson Nano
DAP	$0.74_{\pm 0.00}$	$18.72_{\pm 0.45}$	$5.11_{\pm 0.05}$	$0.43_{\pm 0.00}$	$19.05_{\pm 0.31}$	$5.24_{\pm 0.15}$	$0.45_{\pm 0.01}$	$18.82_{\pm 0.32}$	$5.20_{\pm 0.10}$	$0.44_{\pm 0.02}$	$18.61_{\pm 0.35}$	$5.28_{\pm 0.11}$
IAP	$0.74_{\pm 0.00}$	$18.61_{\pm 0.20}$	$5.11_{\pm 0.03}$	$0.44_{\pm 0.00}$	$18.85_{\pm 0.07}$	$5.28_{\pm 0.13}$	$0.45_{\pm 0.02}$	$18.70_{\pm 0.08}$	$5.24_{\pm 0.09}$	$0.43_{\pm 0.00}$	$18.49_{\pm 0.06}$	$5.19_{\pm 0.11}$
SAE	$0.12_{\pm 0.00}$	$1.11_{\pm 0.13}$	$1.21_{\pm 0.05}$	$0.12_{\pm 0.00}$	$1.35_{\pm 0.09}$	$1.57_{\pm 0.02}$	$0.13_{\pm 0.00}$	$1.66_{\pm 0.07}$	$1.88_{\pm 0.05}$	$0.13_{\pm 0.00}$	$2.29_{\pm 0.07}$	$2.42_{\pm 0.04}$
ESZSL	$0.03_{\pm 0.00}$	$0.81_{\pm 0.09}$	$0.49_{\pm 0.05}$	$0.03_{\pm 0.00}$	$1.08_{\pm 0.10}$	$0.75_{\pm 0.06}$	$0.04_{\pm 0.00}$	$1.40_{\pm 0.15}$	$0.97_{\pm 0.07}$	$0.06_{\pm 0.00}$	$1.89_{\pm 0.05}$	$1.37_{\pm 0.02}$
DEM	$0.83_{\pm 0.05}$	$4.15_{\pm 0.03}$	$9.43_{\pm 0.49}$	$1.54_{\pm 0.12}$	$7.35_{\pm 0.12}$	$13.33_{\pm 0.32}$	$3.11_{\pm 0.14}$	$26.74_{\pm 0.80}$	$24.28_{\pm 1.47}$	$6.25_{\pm 0.23}$	$44.69_{\pm 1.28}$	$40.29_{\pm 2.94}$
f-CLSWGAN	$0.71_{\pm 0.06}$	$3.95_{\pm 0.07}$	$2.02_{\pm 0.12}$	$0.83_{\pm 0.09}$	$4.40_{\pm 0.09}$	$1.98_{\pm 0.09}$	$0.96_{\pm 0.10}$	$5.72_{\pm 0.12}$	$1.79_{\pm 0.05}$	$1.26_{\pm 0.11}$	$7.03_{\pm 0.14}$	$2.55_{\pm 0.13}$

4.2.5 ZSL Methods: Computational Analysis

The previous sections analyzed the performance of the ZSL approaches on various datasets and the consumed time in the inference phase of ZSL approaches. As stated in section 4.2.2, the first phase involves the processing of the image to get its descriptions in a compact format, typically performed using CNNs, and the last phase corresponds to the class prediction task.
However, the analysis of the impact of using a lightweight architecture in ZSL accuracy was not discussed yet. Consequently, this section intends to answer the following two questions: (1) *How much the accuracy varies when using lightweight architectures instead of the de facto standard ResNet101 architecture?*, and (2) *What is the throughput of the ZSL method considering the elapsed time in both the preprocessing phase and the class prediction phase?*.

4.2.5.1 Performance of ZSL methods in different CNN architectures

Regarding the first question, we show in figure 4.6 and figure 4.7 the accuracy of different ZSL approaches when using different CNNs.

As established in section 4.2.3, *ResNet101* is the architecture with the best performance in all assessed datasets. Based on the observation of graph plots in figure 4.6, we can conclude that in the AWA2 dataset, the loss of accuracy is 0.6% when adopting the fastest

CNN architecture on Raspberry Pi 4B: *MobileNetV2*. This value increases to 2.72% if *MobileNet* is adopted. Regarding the generalized setting, the performance drops 0.79% in the case of *MobileNetV2* is used and 1.8% if *MobileNet* architecture is adopted.
For CUB dataset, the results indicate that the loss is 3.19% if *MobileNetV2* is used. This loss value doubles in case of *MobileNet*. In the case of the generalized setting, *MobileNet* surpasses the *MobileNetV2* performance, and the loss is 2% and 2.76%, respectively.
Regarding the SUN dataset, the accuracy drops 3.20% and 6.27% in *MobileNetV2* and *MobileNet*, respectively. For the GZSL setting, the loss value is 2.60% in the case of *MobileNetV2* is used, and 4.04% if *MobileNet* is adopted.
Finally, for the APY dataset, the loss of accuracy is 3.88% in *MobileNetV2* and 1.66% if *MobileNet* is adopted. However, the performance drop is less than 0.6% for *MobileNet* and *MobileNetV2* in the generalized setting.

Similarly, the results obtained through Jetson Nano are depicted in the figure 4.7. Since the fastest architecture is the *MobileNet*, in the case of Jetson Nano, the accuracy drop is slightly higher than in *MobileNetV2*, as stated above. Although the consumed time discrepancy between the two lightweight architectures is 5.46 ms, this does not represent a significant performance drop in general. Therefore, *MobileNetV2* architecture can be considered for a higher accuracy value.

In general, *MobileNetV2* architecture appears to be a good choice as a successor of *ResNet101* since the accuracy drop is less than 4% in all evaluated datasets. This means that using lightweight architectures does not represent a significant performance drop in overall. On the other hand, the accuracy does not necessarily improve with the rising complexity of the model. For example, the results obtained from *Xception* and *EfficientNetB7* architectures are slightly worse when compared to the *MobileNetV2* results in most datasets.

4.2.5.2 Accuracy/Speed trade-off in ZSL methods

In order to provide an answer for the second question, an analysis regarding the accuracy/speed trade-off in ZSL methods was conducted. To this end, we measure the Frame Rate per Second as being $FPS = 1000/(FE_{time} + CP_{time})$, where FE_{time} denotes the elapsed time in feature extracting task, and CP_{time} stands for the consumed time in the class prediction task. This metric allows us to perceive how many class predictions per second are made by the ZSL method.

The results for the AWA2, CUB, SUN, and APY datasets in the six state-of-the-art ZSL methods performed on Raspberry Pi 4B are depicted in figure 4.9. We can observe that the points in the four plotted graphs can be clustered into two distinct groups, where one of the groups contains the lightweight architectures and the other group is composed of the remaining architectures. As expected, the lightweight architectures have a better performance due to their reduced inference time. Even though, the consumed time by these

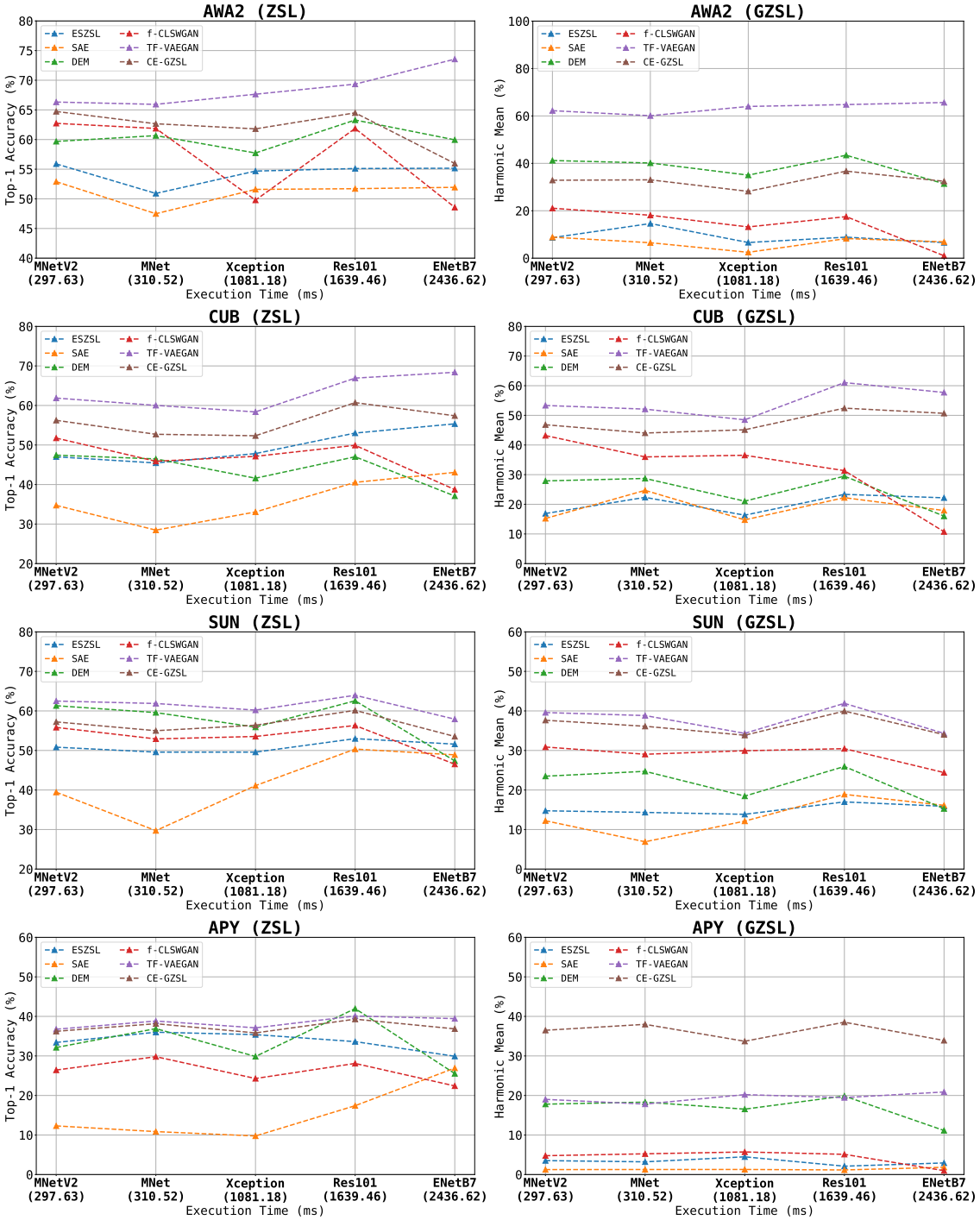


Figure 4.6: **Performance of ZSL methods with respect to the used CNN architecture on Raspberry PI 4B**. Six state-of-the-art ZSL approaches were trained using visual features obtained from CNN architectures with significantly variable complexity. The top-1 accuracy of the evaluated approaches in four ZSL datasets evidences that lightweight architectures do not significantly reduce the performance of ZSL approaches.

networks in feature extraction process when performed on Raspberry Pi 4B is close to 300 ms. Considering the best scenario, disregarding the class prediction time, FPS achieves a maximum value of 3.36 FPS, since the FE_{time} for *MobileNetV2* is 297.63 ms.

Similarly, the results for the ZSL methods performed on Jetson Nano are shown in figure

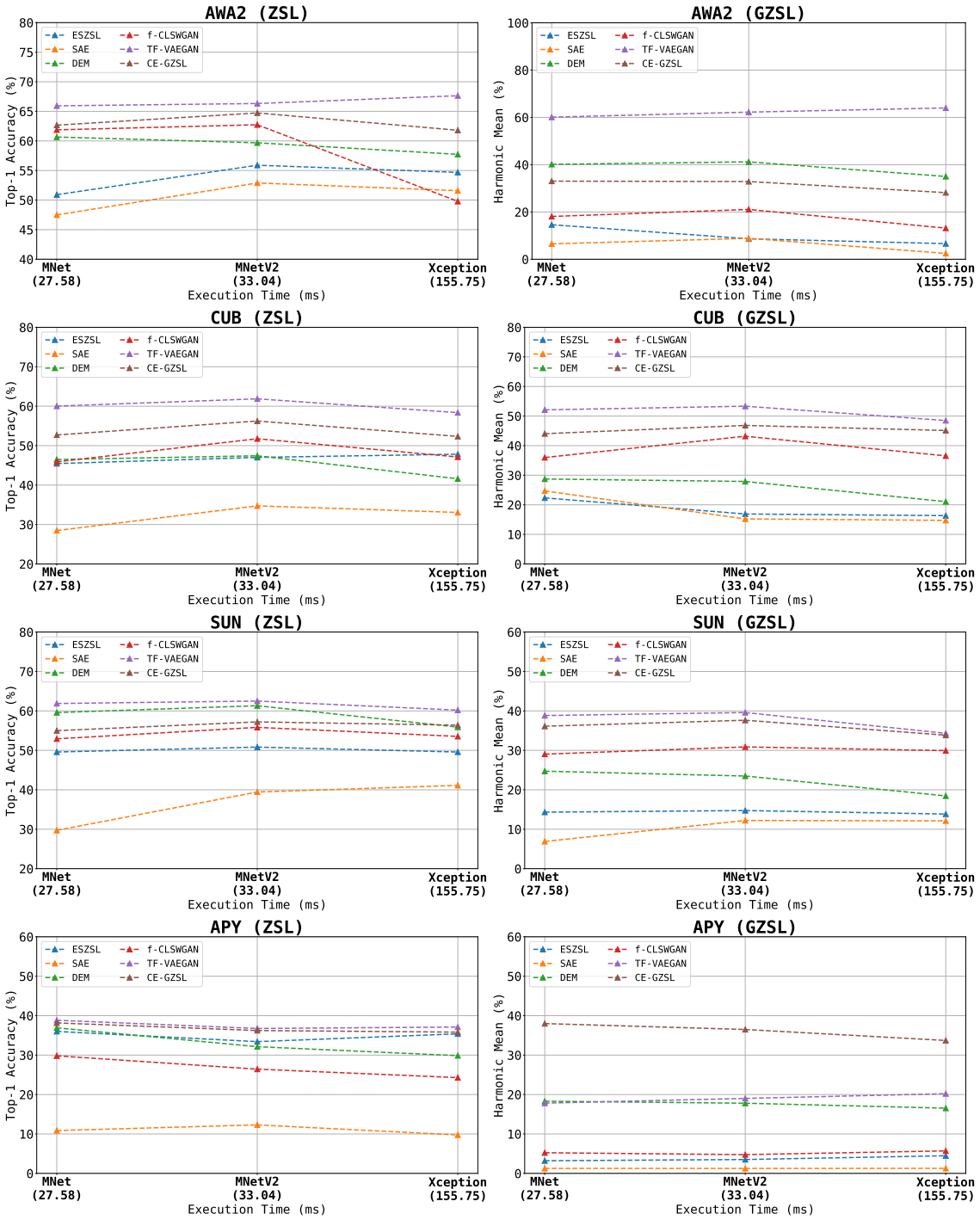


Figure 4.7: **Performance of ZSL methods with respect to the used CNN architecture on Jetson Nano.** Although the MobileNet architecture is fastest than the MobileNetV2, the later can be considered instead of MobileNet architecture for a higher accuracy value.

4.9. Because Jetson Nano is almost ten times faster than Raspberry Pi 4B in the visual features extracting phase, the computed FPS is also ten times superior for the ZSL methods, as we can observe from the analysis of the figure 4.9. This suggests that Jetson Nano has greater performance than Raspberry Pi 4B, as expected.

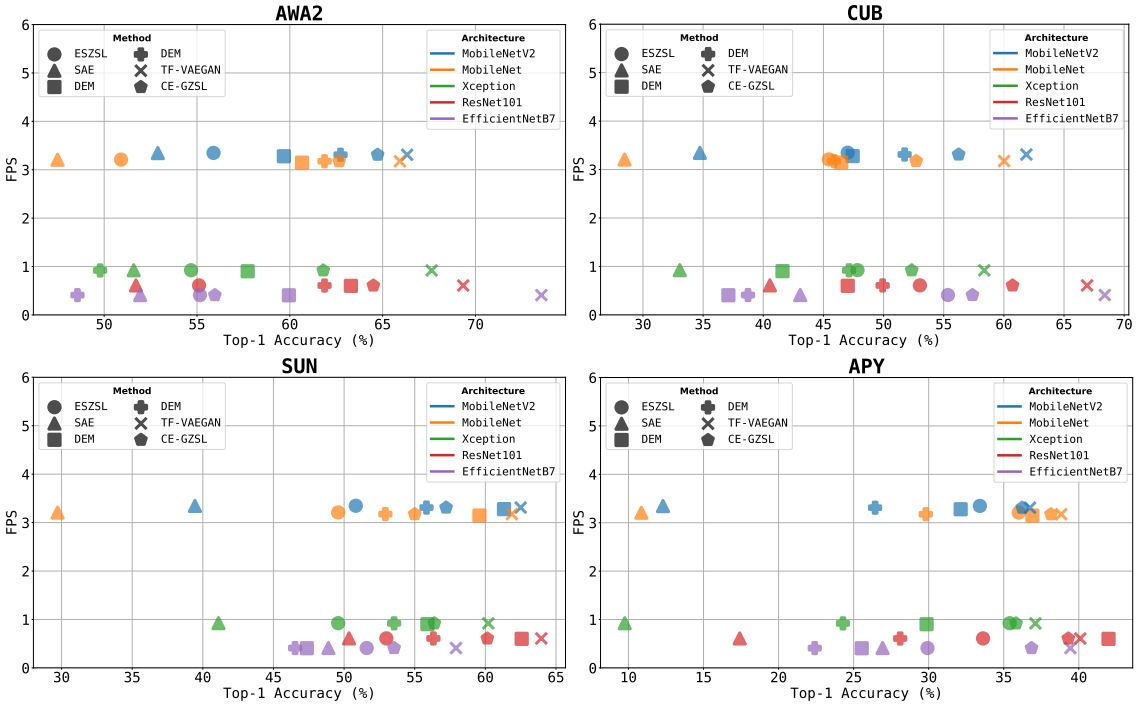


Figure 4.8: **Accuracy/Speed trade-off of ZSL methods with respect to the CNN architecture used when executed on Raspberry PI 4B**. The low FPS values are due to the considerable elapsed time by Raspberry Pi 4B in the visual feature extraction process.

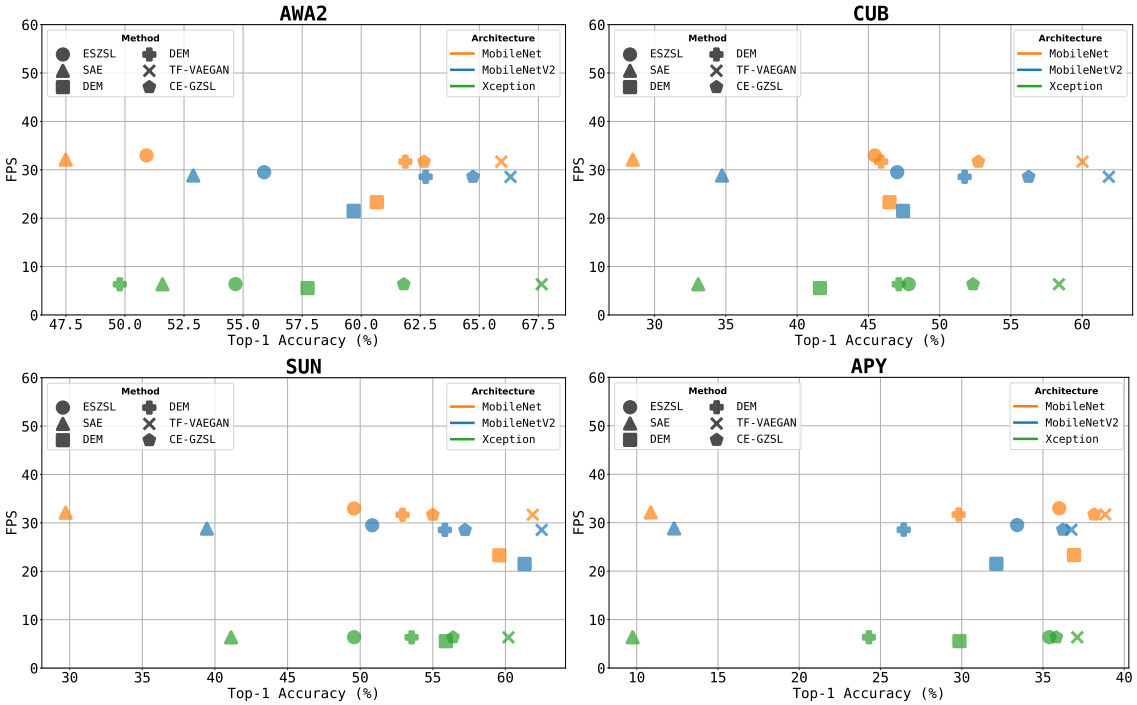


Figure 4.9: **Accuracy/Speed trade-off of ZSL methods with respect to the CNN architecture used when executed on Jetson Nano**. The results evidence that FPS is ten times superior compared to the results on Raspberry Pi 4B.

4.3 Results of the Proposed Strategy

The developed strategy was tested on the CelebA [12] dataset. CelebA is a large-scale face attributes dataset with $202,599$ face images divided by $10,177$ identities. Each image is annotated with 40 binary attributes. Furthermore, we decided to split the dataset into two different splits, each one containing a different number of classes. Table 4.7 provides the detailed statistics for the CelebA dataset.

Table 4.7: Statistics for CelebA [12] dataset in terms of Number of Attributes (Att), Number of Classes in Y_{tr} and Y_{ts}, and Number of Images at Training and Test Time.

			Number of Images			At Training Time	At Test Time	
Dataset	Att	Y	Y_{tr}	Y_{ts}	Total	Y_{tr}	Y_{tr}	Y_{ts}
CelebA(All)	40	10177	$8142 + 814$	1221	202599	143597	34912	24090
CelebA(500)	40	500	$300 + 100$	100	15038	11232	800	3006

For the attribute prediction classifier model, we adopt the VGGFace architecture as the backbone and reconstruct the top layers, as specified in section 3.3.1.
The proposed method was trained as described along the section 3.3, in order to allow the generation of semantic-guided visual feature vectors.
To evaluate the proposed approach, four ZSL methods were selected, including SAE [8], ESZSL [6], DEM [39], and the state-of-the-art TF-VAEGAN generative method. Based on the results presented in table 4.8, the use of the proposed Semantic Guided Attention model (SGAM) increases the classification accuracy by 13.7%, on average, if we consider the most populated 500 classes. On the other hand, the classification accuracy is boosted by 6.45%, on average, if the total of classes is considered. Due to the richness and visual interpretability of attributes, the attention model is very suitable for this dataset.

Table 4.8: **Zero-Shot Learning Results on CelebA** [12]. 500 = Most populated 500 classes. All = All CelebA classes (10177). The results obtained without the proposed SGAM are reported in the column marked with "w/o. SGAM", whereas the results using the SGAM are along the column marked with "w. SGAM". SGAM refers to Semantic Guided Attention Model. The results report top-1 accuracy in %.

	CelebA(500)		**CelebA(All)**	
Method	w/o. SGAM	w. SGAM	w/o. SGAM	w. SGAM
SAE	7.2	**19.1**	1.3	**3.8**
ESZSL	15.7	**32.5**	4.1	**10.2**
DEM	13.1	**25.6**	2.0	**8.5**
TF-VAEGAN	7.9	**21.5**	6.8	**17.5**

However, when we apply the same strategy on traditional ZSL datasets, the results presented in table 4.9 evidence that there is no improvement over the standard features. In case of SAE, the improvement is 0.04%, compared to the result without using discriminative features. On the other hand, in ESZSL the results are worse, with no space for improvement, likewise for DEM and TF-VAEGAN methods. We argue that the proposed method is effective when the annotated attributes are visually interpretative enough for learning a robust attribute prediction model, which inherently allows generating more ac-

curate class-discriminative Grad-CAM maps so that the obtained attribute attention map captures more important regions that enhance the quality of the final semantic-guided feature vector.

Table 4.9: **Zero-Shot Learning Results on AWA2 using the proposed method**. The results obtained without the proposed SGAM are reported in the column marked with "w/o. SGAM", whereas the results using the SGAM are along the column marked with "w. SGAM". SGAM refers to Semantic Guided Attention Model. The results report top-1 accuracy in %.

	AWA2	
Method	w/o. SGAM	w. SGAM
SAE	52.89	**52.93**
ESZSL	55.89	51.94
DEM	59.68	48.07
TF-VAEGAN	66.32	**66.56**

To support this assumption, we provide in figure 4.10 the generated Grad-CAM maps for a horse image. The visual inspection of the figure show that the attribute predictions are quite random and meaningless regarding the horse class. As a result, the obtained AAM has a lot of noise information instead of having discriminative information of the regions related to the attributes. Therefore, the resulting feature vector is not discriminative enough to enhance the method's performance in making accurate predictions.

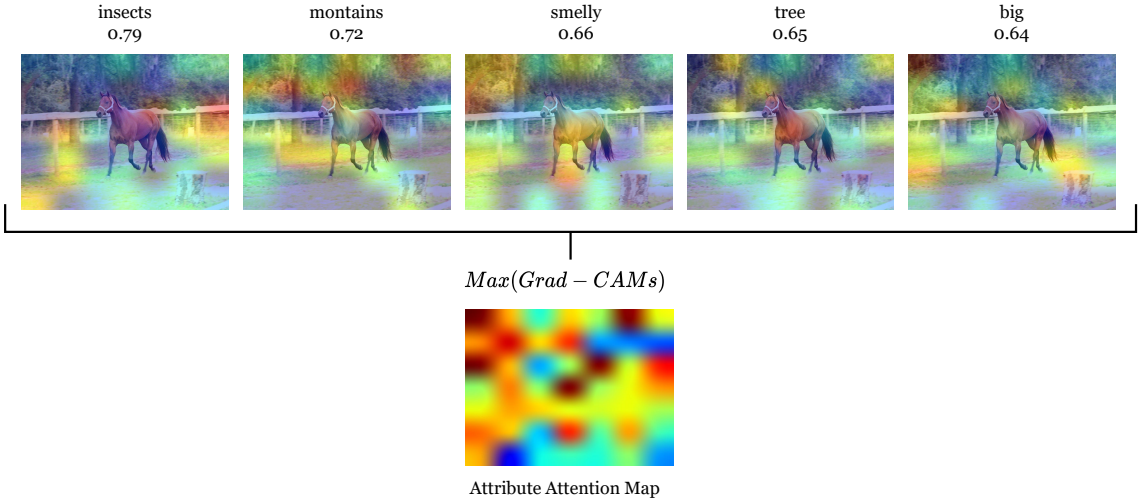


Figure 4.10: **Attribute Attention Map for a horse image**. The predicted attributes are not visually interpretable enough for generating a class-discriminative Grad-CAM. Instead of the generated AAM focusing on the horse class important regions, it highlights other useless parts.

4.4 Discussion

An extensive benchmark regarding the computational performance of the ZSL methods in low-power devices using lightweight architectures was carried out in this chapter. The obtained results showed that using a lightweight architecture, such as *MobileNetV2*, does not significantly reduce overall performance compared to the results on the standard *ResNet101* architecture. As expected, the generative approaches significantly outperform

the other strategies, in particular TF-VAEGAN method. Furthermore, the obtained results evidence that Jetson Nano allows running the ZSL methods in real-time (about 30 FPS), but only when the visual feature extraction models are optimized to run on GPU. Finally, the results in the proposed approach reveal that it can be effective when applied to the datasets with visually interpretable attribute annotations, such as the CelebA dataset.

Chapter 5

Conclusion

The main goal of this work was the development of a ZSL-based approach method capable of recognizing unknown objects using only its textual description. In addition, the developed strategy should be suitable to be embedded into a moving robot using low-power devices.
For this, in chapter 2 an extensive review on ZSL state-of-the-art was carried out in order to identify the status-quo of the ZSL paradigm, as well as the major challenges and the most promising strategies to address the ZSL problem. Also, several ZSL approaches were implemented and evaluated using the unified evaluation protocol that comprehends the assessment of the methods under two different settings: standard ZSL and GZSL.
Although most ZSL works adopt the *ResNet101* architecture as feature extractor network, this type of network is unsuitable for running on low-power devices. Hence, we proposed an evaluation framework for building an extensive benchmark on the impact of using lightweight CNN architectures in the ZSL performance in terms of accuracy/speed. Then, several types of CNN architectures were selected, including lightweight architectures, for evaluating the performance on four benchmark datasets using six state-of-the-art approaches. The used testbed devices included a Raspberry Pi 4B and a Jetson Nano. The results presented in chapter 4 allowed us to conclude that when a lightweight architecture is used, the impact on the ZSL accuracy is not significant. Moreover, Jetson Nano outperforms the Raspberry Pi 4B in terms of processing speed, indicating that it can be considered a suitable device to perform the ZSL task. However, to take advantage of the GPU capabilities, it is crucial optimizing the models to effectively run on GPU, as stated in section 3.2.
Finally, beyond the robust computational performance analysis described above, we developed a ZSL-based approach to combine two promising concepts under the scope of ZSL research. The idea was to fuse a Semantic-Guided Attention Model with a generative model to take the best of both worlds. The results evidenced that our proposed method is valid when applied to the datasets with visual interpretable attribute annotations. As a result, the model does not improve the performance when applied to the traditional ZSL datasets, such as AWA2.
To conclude, our main objectives were fulfilled successfully, and we are convinced that our open-source evaluation framework will be helpful in the ZSL research area.

5.1 Contributions and Achievements

The major contribution of our work is an extensive benchmark across four widely known ZSL datasets regarding the accuracy and processing speed of state-of-the-art ZSL meth-

ods using visual features obtained from lightweight architectures. Furthermore, we developed an open-source evaluation framework for analyzing the accuracy/speed trade-off in the problem of ZSL (`https://github.com/CristianoPatricio/zsl-methods`). Moreover, we provided a comparative analysis of the processing speed of ZSL algorithms using lightweight architectures when run in different low-power devices.
The results on the proposed SGAM method demonstrated that by learning discriminative visual features, it is possible to improve the classification accuracy of the ZSL methods, however depending on the attribute properties of the used dataset.
Additionally, the work described in this dissertation has allowed the writing of a scientific paper describing a concise benchmarking on the computational performance of ZSL in low-power devices, allowing to perceive how the accuracy of state-of-the-art approaches is impacted when lightweight architectures are adopted.

5.2 Future Work

The work described in this dissertation can be complemented with the analysis of other lightweight architectures and also a broader range of ZSL methods to find the maximum balance between accuracy and inference speed. On the other hand, we believe that the proposed ZSL method can be improved in order to be employed in datasets that lack visually interpretable attribute annotations.

Bibliography

[1] Y. Xian, T. Lorenz, B. Schiele, and Z. Akata, "Feature generating networks for zero-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5542–5551. xiii, xv, 3, 4, 6, 21, 22, 23, 26, 32, 33, 36, 44, 45, 51

[2] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata, "Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 9, pp. 2251–2265, 2018. xiii, xix, 8, 25, 28, 29, 30, 31, 32, 37, 39, 45, 46, 47, 49, 50

[3] R. Socher, M. Ganjoo, C. D. Manning, and A. Ng, "Zero-shot learning through cross-modal transfer," *Advances in Neural Information Processing Systems*, vol. 26, pp. 935–943, 2013. xiii, 11, 12, 32, 33

[4] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, M. Ranzato, and T. Mikolov, "Devise: A deep visual-semantic embedding model," in *Advances in Neural Information Processing Systems*, 2013, pp. 2121–2129. xiv, 11, 12, 13, 14, 23, 30, 32, 33

[5] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid, "Label-embedding for attribute-based classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 819–826. xiv, 15, 16, 27, 32, 33

[6] B. Romera-Paredes and P. Torr, "An embarrassingly simple approach to zero-shot learning," in *International Conference on Machine Learning*, 2015, pp. 2152–2161. xiv, 16, 17, 18, 23, 31, 32, 33, 36, 45, 51, 57

[7] Y. Xian, Z. Akata, G. Sharma, Q. Nguyen, M. Hein, and B. Schiele, "Latent embeddings for zero-shot classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 69–77. xiv, 17, 18, 23, 27, 32, 33

[8] E. Kodirov, T. Xiang, and S. Gong, "Semantic autoencoder for zero-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3174–3183. xv, 9, 19, 20, 21, 27, 28, 31, 32, 33, 36, 45, 51, 57

[9] A. Mishra, S. Krishna Reddy, A. Mittal, and H. A. Murthy, "A generative model for zero shot learning using conditional variational autoencoders," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 2188–2196. xv, 21, 24, 25, 26, 27, 29, 32

[10] Y. Xian, S. Sharma, B. Schiele, and Z. Akata, "f-vaegan-d2: A feature generating framework for any-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 275–10 284. xv, 7, 21, 25, 26, 28, 32, 33, 40

[11] NVIDIA, "Nvidia deep learning tensorrt documentation," [Accessed: Jun 1, 2021]. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#overview xv, 38, 39

[12] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. xvi, xx, 43, 44, 57

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. xix, 12, 13, 28, 31, 33

[14] IBM, "Artificial intelligence (ai)," [Accessed: Jan 16, 2021]. [Online]. Available: https://github.com/chichilicious/embarrsingly-simple-zero-shot-learnin 1

[15] W. Wang, V. W. Zheng, H. Yu, and C. Miao, "A survey of zero-shot learning: Settings, methods, and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–37, 2019. 1, 3, 5, 6, 28

[16] A. P. Masucci, A. Kalampokis, V. M. Eguíluz, and E. Hernández-García, "Wikipedia information flow analysis reveals the scale-free architecture of the semantic space," *PloS one*, vol. 6, no. 2, p. e17333, 2011. 3

[17] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008. 3

[18] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth, "Describing objects by their attributes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 1778–1785. 5, 28, 30, 37, 45, 46

[19] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995. 5

[20] Z. Akata, S. Reed, D. Walter, H. Lee, and B. Schiele, "Evaluation of output embeddings for fine-grained image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2927–2936. 5, 23

[21] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in Neural Information Processing Systems*, vol. 26, pp. 3111–3119, 2013. 6, 12

[22] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1532–1543. 6

[23] C. H. Lampert, H. Nickisch, and S. Harmeling, "Learning to detect unseen object classes by between-class attribute transfer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 951–958. 7, 28, 30, 31, 32, 37, 45, 46

[24] M. Palatucci, D. Pomerleau, G. E. Hinton, and T. M. Mitchell, "Zero-shot learning with semantic output codes," *Advances in Neural Information Processing Systems*, vol. 22, pp. 1410–1418, 2009. 10, 31

[25] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013. 12

[26] J. Deng, A. Berg, S. Satheesh, H. Su, A. Khosla, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015. 12

[27] M. Norouzi, T. Mikolov, S. Bengio, Y. Singer, J. Shlens, A. Frome, G. S. Corrado, and J. Dean, "Zero-shot learning by convex combination of semantic embeddings," *arXiv preprint arXiv:1312.5650*, 2013. 13, 14, 30, 32, 33

[28] S. Direct, "Convex combination," [Accessed: Dec 15, 2020]. [Online]. Available: https://www.sciencedirect.com/topics/mathematics/convex-combination 14

[29] ——, "Cosine similarity," [Accessed: Dec 15, 2020]. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/cosine-similarity 14

[30] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large margin methods for structured and interdependent output variables," *Journal of machine learning research*, vol. 6, no. Sep, pp. 1453–1484, 2005. 15

[31] Wikipedia, "Sylvester equation," [Accessed: Dec 17, 2020]. [Online]. Available: https://en.wikipedia.org/wiki/Sylvester_equation 20

[32] E. Schonfeld, S. Ebrahimi, S. Sinha, T. Darrell, and Z. Akata, "Generalized zero-and few-shot learning via aligned variational autoencoders," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8247–8255. 21

[33] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," *Advances in Neural Information Processing Systems*, vol. 28, pp. 3483–3491, 2015. 24

[34] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. 25

[35] Y. Zhu, J. Xie, Z. Tang, X. Peng, and A. Elgammal, "Semantic-guided multi-attention localization for zero-shot learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 943–14 953. 26, 32, 40

[36] Y. Fu, T. M. Hospedales, T. Xiang, and S. Gong, "Transductive multi-view zero-shot learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 11, pp. 2332–2345, 2015. 27

[37] S. Chandhok, "Zero-shot learning: An introduction," [Accessed: Dec 23, 2020]. [Online]. Available: https://www.learnopencv.com/zero-shot-learning-an-introduction/ 27

[38] G. Dinu, A. Lazaridou, and M. Baroni, "Improving zero-shot learning by mitigating the hubness problem," *arXiv preprint arXiv:1412.6568*, 2014. 27, 28

[39] L. Zhang, T. Xiang, and S. Gong, "Learning a deep embedding model for zero-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2021–2030. 28, 36, 40, 45, 51, 57

[40] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-ucsd birds-200-2011 dataset," 2011. 28, 30, 37, 45, 46

[41] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li, "Novel dataset for fine-grained image categorization: Stanford dogs," in *Proceedings of the CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, 2011. 28

[42] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009. 28

[43] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar, "Attribute and simile classifiers for face verification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2009, pp. 365–372. 28

[44] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145–175, 2001. 28

[45] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Proceedings of the Indian Conference on Computer Vision, Graphics & Image Processing*, 2008, pp. 722–729. 28

[46] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 633–641. 28, 30, 37, 45, 46

[47] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017. 28

[48] W.-L. Chao, S. Changpinyo, B. Gong, and F. Sha, "An empirical study and analysis of generalized zero-shot learning for object recognition in the wild," in *European Conference on Computer Vision*. Springer, 2016, pp. 52–68. 29

[49] Wikipedia, "Harmonic mean," [Accessed: Dec 22, 2020]. [Online]. Available: https://en.wikipedia.org/wiki/Harmonic_mean 30

[50] S. Bharadwaj, "embarrsingly-simple-zero-shot-learning," https://github.com/chichilicious/embarrsingly-simple-zero-shot-learning, 2018. 31

[51] S. Narayan, A. Gupta, F. S. Khan, C. G. Snoek, and L. Shao, "Latent embedding feedback and discriminative features for zero-shot classification," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. 36, 37, 45, 51

[52] Z. Han, Z. Fu, S. Chen, and J. Yang, "Contrastive embedding for generalized zero-shot learning," *arXiv preprint arXiv:2103.16173*, 2021. 37, 40, 45, 49, 51

[53] A. Paul, "Feature-generating-networks-for-zsl," https://github.com/akku1506/Feature-Generating-Networks-for-ZSL, 2018. 37

[54] A. Gupta, "tf-vaegan," https://github.com/akshitac8/tfvaegan, 2020. 37

[55] Hanzy1996, "Ce-gzsl," https://github.com/Hanzy1996/CE-GZSL, 2021. 37

[56] B. Zhao, Y. Fu, R. Liang, J. Wu, Y. Wang, and Y. Wang, "A large-scale attribute dataset for zero-shot learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0. 37, 45, 46

[57] Z. Liu, X. Zhang, Z. Zhu, S. Zheng, Y. Zhao, and J. Cheng, "Taking modality-free human identification as zero-shot learning," *arXiv preprint arXiv:2010.00975*, 2020. 40, 43

[58] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 618–626. 40, 41, 42, 43

[59] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2921–2929. 42