THESIS

GAN YOU TRAIN YOUR NETWORK

Submitted by

Venkata Sai Sudeep Pamulapati

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2022

ABSTRACT

GAN YOU TRAIN YOUR NETWORK

Zero-shot classifiers identify unseen classes — classes not seen during training. Specifically, zero-shot models classify attribute information associated with classes (e.g., a zebra has stripes but a lion does not). Lately, the usage of generative adversarial networks (GAN) for zero-shot learning has significantly improved the recognition accuracy of unseen classes by producing visual features on any class. Here, I investigate how similar visual features obtained from images of a class are to the visual features generated by a GAN. I find that, regardless of metric, both sets of visual features are disjointed. I also fine-tune a ResNet so that it produces visual features that are similar to the visual features generated by a GAN — this is novel because all standard approaches do the opposite: they train the GAN to match the output of the model. I conclude that these experiments emphasize the need to establish a standard input pipeline in zero-shot learning because of the mismatch of generated and real features, as well as the variation in features (and subsequent GAN performance) from different implementations of models such as ResNet-101.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Nathaniel Blanchard for his continuous support throughout my masters studies. I would like to thank my co-advisor Dr. Ross Beveridge for his insights and positive feedback on my research. I also want to thank my advisory committee member Dr. Emily King for being a part of this committee. I want to thank my parents, Pamulapati Subbarao, Pamulapati Rama Devi and my brother Pamulapati Tarun Sai for their support throughout my masters degree. I would like to thank Chaitanya Tripuraneni for his guidance.

DEDICATION

I would like to dedicate this thesis to my teachers and parents.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Machine learning has made significant strides in the recent years, especially in the field of computer vision. The invention of Convolutional Neural Network (CNNs) has empowered researchers to solve many complex real world problems like object recognition and detection [1, 2], semantic segmentation [3, 4], image super resolution [5, 6] etc. For instance, ResNet [7], a state of art Convolutional Neural Network achieved a top-5 error of 3.6% on ImageNet dataset where as humans could only achieve a top-5 error rate of 5.1% [8]. It took approximately 30 years after the introduction of Back propagation, a technique to train neural networks, to come up with a model that could surpass human performance in object recognition. One primary reason for this is because convolutional neural networks are data thirsty. The introduction of ImageNet [9], a benchmark dataset to evaluate and train CNNs in the year 2009, has expedited the research in the field of object recognition. ImageNet has over 3.2 million images belonging to 1000 different classes. To annotate these images, authors of ImageNet used Amazon Mechanical Turk, where each image is assigned a label when it receives convincing number of votes. The growing research in the field of computer vision has led to the development of complex algorithms with billions of parameters, which is enabling us to solve many complex, real-world challenges. This growing number of parameters in machine learning models makes them further dependent on large datasets.

Annotating and labeling images is tedious and time consuming, especially when we consider more complex problems like semantic segmentation and object localization. For example, it took four years to annotate the ImageNet dataset, with the average worker annotating 50 images per minute. The number of newly emerging objects are also growing and it is hard to keep track of them. It is important to come up with annotation approaches that require fewer data samples and limited human oversight. This would also help users develop machine learning applications quickly. For example, during the early stages of the COVID-19 pandemic, developing a computer vision algorithm that would learn to recognize COVID-19 patients based on training on X-Ray images of patients with similar inflammatory diseases would have been really helpful. A technique that would learn from the existing data and translate its learning to a new task would be a perfect solution to be problem described before. Applications like this is where zero-shot learning (ZSL) comes to use. The applications of zero-shot learning in computer vision include recognizing unseen emotions [10], information retrieval systems [11, 12], new action recognition [13, 14], etc. Zero shot learning is also used in Natural Language Processing tasks like: Zero-Shot Entity Linking [15], sentence embedding [16], and language translation [17].

Zero-shot learning is a domain of machine learning where networks are trained to classify instances of classes that are not seen during training. An example can be seen in Fig. 1.1. We have a network trained on images Puma and we want the network to classify a black panther with the help of some side information, like attributes. Examples of attributes can be seen in Fig. 1.2. Here, both the classes of images are similar, so it would achievable to use the results from Puma to recognize a black panther. Let's assume we have two disjoint sets of classes of images called *train set* and *test set*. Any proposed algorithm is trained on the train set. Since, the network has seen images of classes in the train set, the classes in the train set are commonly referred as seen classes. The performance of the network is evaluated on the test set; because the classes of images in the test set are not seen during training, they are called *unseen classes*. Any dataset used for zero-shot learning contains images and their corresponding label classes. In addition to images for each class, attributes are also provided. The zero-shot learning algorithm can make use of images, labels, and attributes of seen classes. In addition, it is permitted make use of attributes of unseen classes.

The performance of any zero-shot learning algorithm is evaluated on two different tasks of zero-shot learning. These two different tasks are generalized zero-shot Learning and conventional zero-shot learning. In conventional zero-shot learning, the classes in the train set and the test set are disjoint. However, in generalized zero-shot learning, there is a overlap between the train and test set classes (i.e, the test set contains images from both the seen and unseen classes). The train set remains the same in both the cases. The difference between conventional zero-shot learning

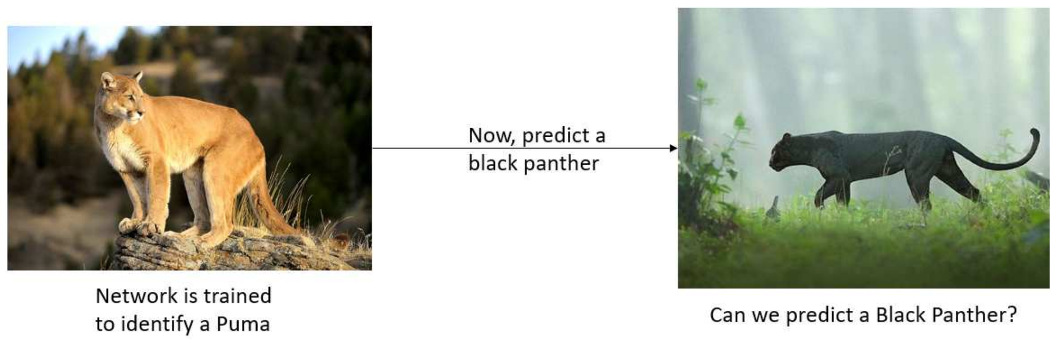
Now, predict a black panther
Network is trained to identify a Puma
Can we predict a Black Panther?

Figure 1.1: Example for Zero Shot Learning.

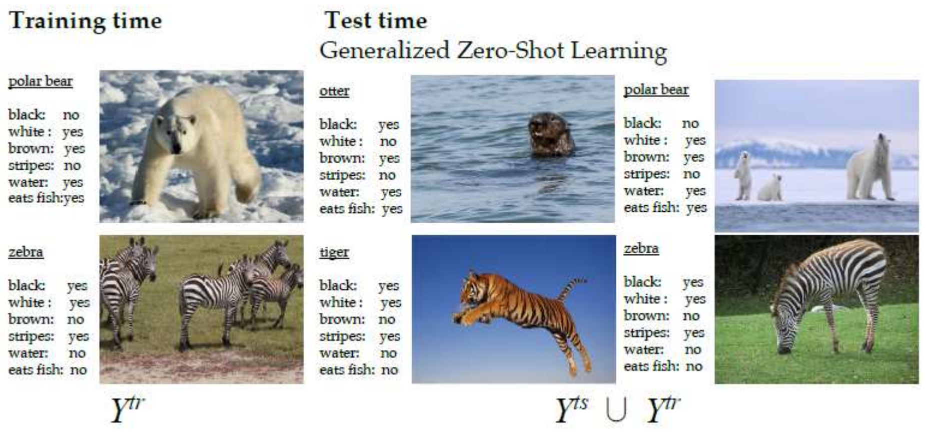

Figure 1.2: Difference between Zero Shot Leaning and Generalized Zero Shot Learning setting. For both the settings at the train time, images and attributes of train set (Y^{tr}) are available. At the test time, in Generalized Zero-Shot Learning setting, the algorithm is evaluated on the images from both the train and test set ($Y^{ts} \cup Y^{tr}$). [19]

and generalized zero-shot Learning is illustrated in Fig. 1.2. To train a zero-shot learning model, attributes of seen and unseen classes are provided in addition to the images. Attributes are textual descriptors of a class. In the CUB dataset [18], which contains images of birds, attributes like Nape color, Bill shape, Throat Color, etc. are provided.

Introduction of Generative Adversarial Networks (GANs) has revolutionized zero-shot learning. Zero-shot learning algorithms which did not use GANs suffer from training data imbalance between the seen and unseen classes and this in turn leads to poor performance on generalized zero-shot learning tasks. In order to overcome this data imbalance issue, GANs are used to generate visual features of unseen classes using their attributes. Yongqin Xian, et al. [20] demonstrated that using such an approach helped to significantly boost the performance on five different datasets. Even-though GANs achieved superior accuracy, there are many unanswered questions in this approach. In my thesis I answered some of the questions. These are the questions I answered in my thesis:

- The degree of similarity between the visual features produced by GANs and ResNet.
- A novel student-teacher approach to improve the similarity of both sets of visual features.
- The effect of different ResNet visual features on the performance of GANs in zero-shot learning.

My research involves understanding the similarity between features generated by GANs and features extracted from images using a ResNet101. I also tried to improve the similarity of both sets of visual features using a novel student-teacher network architecture. In these experiments, I also found that there are different versions of ResNet101 in use and used these different versions of ResNet101 to train zero-shot learning algorithms. I found out that changing the version of ResNet101 significantly affected the performance of the zero-shot learning algorithm, emphasizing the importance of establishing a standard pipeline to extract visual features from images in zero-shot learning.

Chapter 2

Previous Works

There has been a rapid increase in the number of zero-shot learning algorithms proposed in the last few years. The algorithms proposed have developed from simple linear learning compatibility methods to complex Generative Adversarial networks. Some of the initial works, like Attribute Label Embedding (ALE) [21], viewed zero-shot learning as a label embedding problem. This involved projecting the images into the embedding space. The authors introduced a compatibility function which, when given an image, measures the compatibility with a label embedding. The idea here is that the label embedding of a class would have a high degree of compatibility with its corresponding class images when compared to others. Similarly, DeViSe [22] projected the intermediate representation of an image from a neural network into the semantic space. Semantic features for each class were obtained using a skip gram style network, and the neural network used to obtain visual features from an image was pretrained on ImageNet [9] dataset. The network used for projecting visual features onto a semantic space was trained using a combination of dot-product similarity and hinge rank loss. A Semantic Auto Encoder (SAE) was used in [23] where the latent space of the auto encoder represents the semantic space. The auto encoder was trained to project visual features into semantic space and then effectively reconstructed them back from the semantic space. In [24], authors similarly mapped the images into the semantic space using a neural network model. Classifiers were then used to classify the predicted semantic features at test time. An outlier detection model was adopted in addition to the classifiers, given that a classifier would lean towards assigning an image to one of the seen classes. If an outlier was detected, a separate classifier was used to classify the image into one of the unseen classes. All the methods above used a globally linear compatibility function to map from image space to semantic space. Alternatively, [25] built a nonlinear compatibility function which is piecewise linear to map between image and embedding space. Instead of mapping features in one space to another, the authors mapped both the visual and embedding features into a different multi-dimensional feature space. [26] explained that class attribute information extracted from online text corpora or sources is noisy, and proposed an objective function that effectively reduces the noise in the attributes.

[27], [28], [29] also embedded either or both the semantic and visual features into a different intermediate space. Ziming Zhang el al in [27] used two different learned functions to project image and class embeddings into a common space in which features were a mixture of seen class proportions. When an unseen class image was presented at a test time, it was projected onto the common space and compared with the projections obtained from unseen class attributes using inner product. In [28], instead of projecting the images into the semantic space, researchers used a classifier to classify the image. Top "t" classifications for the image were taken, and the attribute prediction of the image was represented as the combination semantic embeddings weighted by their corresponding predicted probabilities. Then, the calculated semantic embedding was assigned a label based on the ground truth class embeddings of the unseen class. Instead of projecting images into a different space, [30] took an image attribute pair as input and predicted their consistency. Li Zhang, et al. in [31] argued that projecting images onto semantic space or an intermediate space would shrink the variance in the data and that this, in turn, would aggravate the hubness problem. Instead, the authors projected semantic features into visual feature space using deep neural network-based embedding model (DEM), which also allows fusion of semantic features from different sources. To improve the visual-semantic feature of space mapping, dual visual-semantic mapping paths (DMap) were put forward in [32], which would also generate a new semantic space highly correlated to the visual feature space. Rather than projecting the visual features into semantic space, [33] utilized three different gaze embeddings which were extracted from the gaze data, and projected the visual features into the gaze embedding space.

The works discussed above do not use any information from the unseen classes and it is called as inductive zero-shot learning setup. However, in the transductive zero-shot learning setup, unseen class visual features (without the labels) or attributes are permitted to be used during the training. In this setting, usage of unseen class information is allowed to be used without accessing their associated class labels. In the new transductive settings, [19] demonstrated how to improve the performance of ALE algorithm discussed above by using the label propagating procedure as discussed in [34]. In the [35], the authors prove that transductive setup would also help to prevent the domain shift problem. Because both the seen and unseen class are unrelated, their distributions differ. To overcome this domain shift problem, authors in [35] used a single visual prototype for each of the unseen classes. [36] modelled class distributions of seen and unseen classes using exponential family distributions The parameters of the modelled distribution were contained on the respective class attribute embeddings using a linear/nonlinear regression model.

The advent of Generative Adversarial Networks (GANs) has revolutionized the domain of zero-shot learning. [37] proposed a deep generative model for zero-shot learning where an attribute-specific latent space is learned. The parameters of the latent space are controlled by the output of a trainable deep neural network. The latent space distribution acts as a prior to a Variational Auto Encoder (VAE). At the test time, the latent space distributions of the unseen classes are extracted from their class attributes. Given an unseen class image at the test time, the image is mapped to the latent space using VAE, and the corresponding attributes are predicted by maximizing the lower bound of the VAE. Authors in [38] used a Conditional GAN (CGAN) to generate semantically same and semantically compound samples. These generated samples increase the diversity in training data and help in discriminating overlapping classes. In [39], zero-shot learning is converted into a simple classification problem by generating visual features of classes from text descriptions using a GAN. Once trained, GAN can be used to generate visual features of unseen classes from their textual descriptions. Similar to [39], [20] also generates visual features of any class by using attributes of the class. More details about this work can be found in the background section.

Chapter 3

Background

To train a model in zero-shot learning, we have two sets of data available to us (i.e. images label pairs and class wise attributes). Different zero-shot learning methods can be classified based on the way both these sets are used. We will also have a look at the semantics of different datasets used to evaluate the performance of any zero-shot learning algorithms in this section.

3.1 Datasets

There are four major datasets that are used to evaluate the performance of any zero-shot learning algorithm. These datasets are: CUB, FLO, SUN, AWA2. These four datasets contain different classes of images. CUB dataset contains images of 200 bird species with a total of 6033 images [18]. Annotations like bounding boxes, rough segmentations, and attributes are provided for each image. Attributes are provided at the class level and are collected using Amazon Mechanical Turk. For each class, 25 visual attributes are collected. The FLO dataset contains images of 102 different flower categories with each category having between 40 to 258 images [40]. The SUN dataset contains images of natural scenes. The dataset contains 130,519 images classified in 899 classes [41]. The AWA2 dataset has consists of 37322 images of 50 animals classes. Attributes are collected based on how humans associate an attribute to a clas [42].

3.2 Generative Adversarial Networks

Generative Adversarial Networks (often termed as GANs, for short) are a type of generative algorithm used to generate data in deep learning. GANs are introduced by Ian J. Goodfellow, et al. in [43]. GANs try to find patterns in the input data in such a way that the model can be used to generate new data examples that may look like they have been drawn form the original dataset. GANs are an example of unsupervised learning algorithms. Even though GANs are unsupervised learning algorithms, they frame their task as a supervised learning algorithm. A GAN has two

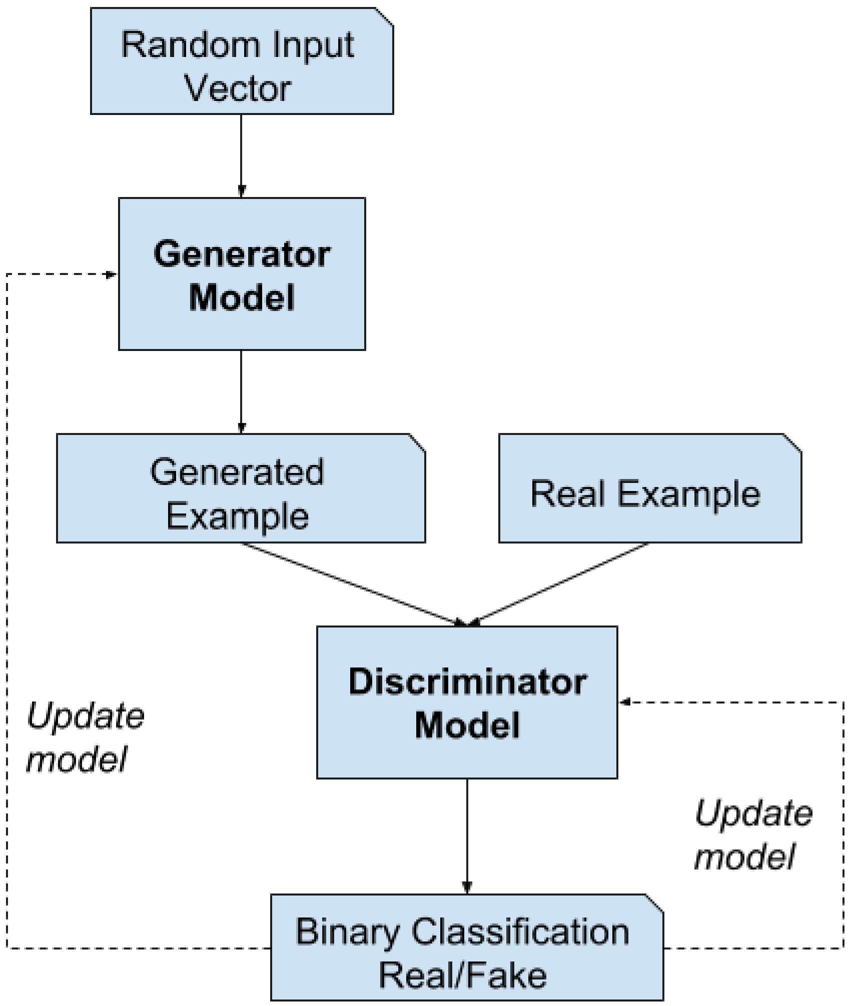

Figure 3.1: General Representation of a Generative Adverserial Network (GAN). Image taken from [44].

components: the generator and the discriminator. The generator creates data from the input and the discriminator tries to distinguish if the generated data is real or fake. Then, both the models are trained together in a zero-sum game. The goal of the generator is to fool the discriminator. At the same time, the discriminator aims to detect the samples generated by the generator as fake. Therefore, the generator and discriminator are working against each other in a adversarial manner. GANs have been increasingly used for several deep learning tasks like image super resolution and image compression. A block diagram of a GAN can be seen in 3.1.

3.3 Methodology

My research is based upon F-CLSWGAN [20]. This section explains the function and design aspects of the algorithm. F-CLSWGAN is one the earliest works that used GANs in zero-shot learning. The core idea behind using GANs is to project the data from attribute space to visual features space. In simple terms, using GANs can be seen as generating visual features using attributes. The generator $G : Z \times C \rightarrow X$ is a conditional generator which takes random Gaussian Noise Z and class embedding C of class y as input and predicts CNN-like image features $\tilde{x} \in X$ of class y. The noise Z promotes variations in the generated visual features. The CNN visual features learned by generator G are conditioned on seen class embedding or attributes. This kind of approach would solve the data imbalance problem between seen and unseen classes as we can generate any number of visual features using GAN. To encourage the generator G to generate discriminative visual features, a classifier K is used in the training process. This training process of the generator involves a classification loss in addition to the negative log likelihood loss, which is used to train a generator of a GAN. The classifier K is trained on the visual features of seen classes. The role of classifier K in the training process is to force the Generator G to generate visual features that are discriminative. The idea behind this is since we are generating visual features that are discriminative by a classifier trained on ResNet visual features, the vice-versa holds true i.e. the ResNet visual features will be classified by a classifier trained on generated visual features. The discriminator is trained to distinguish whether the generated visual features are real or fake. (The authors refer to the algorithm as F-CLSWGAN.) The architecture of F-CLSWGAN can be seen in Fig. 3.2 and the different terms in the figure are explained in table 3.1.

Once the network learns to generate visual features of seen classes based on the class embedding, it generates visual features $\tilde{x}$ of unseen classes using their corresponding class attributes. The visual features of unseen classes are used to train a discriminative classifier. The goal of the discriminative classifier is to classify the visual features of images extracted using a ResNet at the test time. The authors tried two different networks: GoogLeNet [45] and ResNet [7] to extract

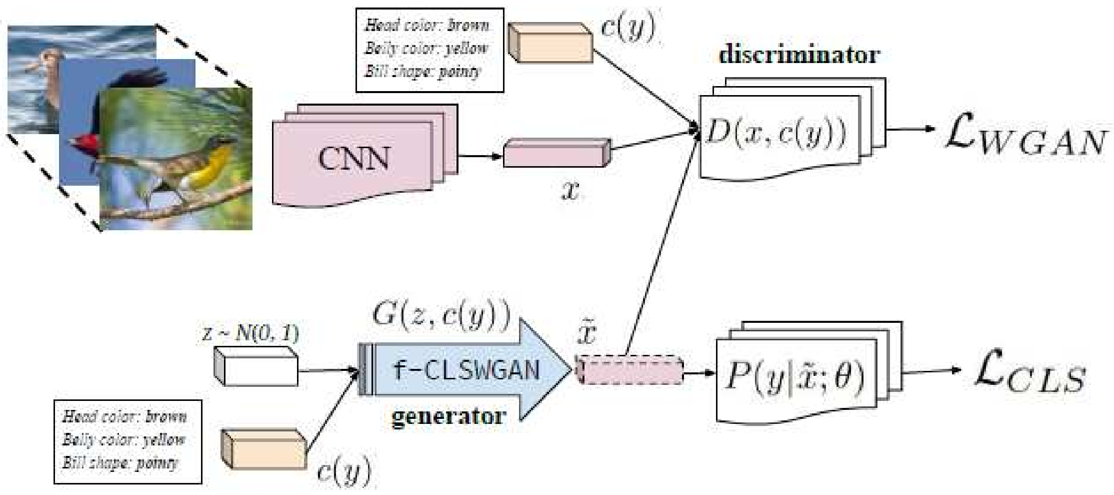

Figure 3.2: F-CLSWGAN architecture. Image taken from [20]

Table 3.1: Different terms in Fig. 3.2

$\mathcal{L}_{WGAN}$	Wasserstein loss
$\mathcal{L}_{CLS}$	Classification loss from classifier k
$c(y)$	Attributes of class y
$G(z, c(y)$	Generator
$D(x, c(y)$	Discriminator
z	random noise
$\tilde{x}$	Generated visual features
x	Visual features extracted from images using ResNet-101

visual features of images at the test time. The authors demonstrated that using a ResNet at a feature extractor yielded better zero-shot learning performance. Visual features were extracted from the pen-ultimate layer of a ResNet. These visual features are often referred to as *ResNet visual features*.

Chapter 4

Experiments

In this chapter, we will be discussing all the experiments performed in my research.

4.1 Similarity Metrics

F-CLSWGAN achieved 69% accuracy on CUB dataset in a conventional zero-shot learning setting. Even though the authors demonstrated state of art accuracy on zero-shot learning, they did not provide information about the similarity between the visual features generated by F-CLSWGAN and the visual features extracted from the images using ResNet. This section explains different experiments I performed to measure the similarity between both sets of visual features. The similarity metrics used in the experiments are: cosine similarity and Pearson's correlation. Before discussing the experiments, it is important to explain how the experiments are conducted. We have two sets of data for each and every class: visual features extracted from the images using ResNet (ResNet visual features) and visual features generated by F-CLSWGAN (GAN visual features).

In statistics, Pearson's correlation coefficient *r* is used to determine whether a linear correlation exists between two sets of data (Eq. 4.1). In this experiment, I wanted to understand if there is any linear correlation between both sets of visual features. From Table 4.2 and Table 4.1 we found that both sets of visual features do not have any linear relationship. To compute Pearson's correlation coefficient all the ResNet visual feature of a particular class is compared with the mean GAN visual feature of that class. Mean GAN visual feature is obtained by averaging all the GAN visual features of that class.

$$r = \frac{\sum(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum(x_i - \overline{x})^2(y_i - \overline{y})^2}} \tag{4.1}$$

Cosine similarity is a metric used to understand how similar two vectors are. It is a dot product between two vectors which are normalized (Eq. 4.2). Using cosine similarity, we can also

Table 4.1: Average inter and intra-class cosine similarity between GAN and ResNet visual features of seen classes

Dataset	Cosine Similarity		Pearson's Correlation	
	Inter-class	Intra-class	Inter-class	Intra-class
CUB	0.55	0.40	0.01	0.01
FLO	0.56	0.40	0.02	0.01
AWA2	0.50	0.33	0.01	0.02

understand the angle between the two vectors. From the experiments, I found out that the average interclass cosine similarity found between the same class visual features is 0.5. The cosine similarity between the visual features can be seen in 4.2 and Table 4.1. We can also see that the inter- (similarity between the same class ResNet and GAN visual features) and intra- (similarity between the different class ResNet and GAN visual features) class cosine similarity of the classes are similar. In an ideal scenario, we want the inter-class cosine similarity to be significantly greater than intra-class cosine similarity. To compute cosine similarity metrics, a GAN visual feature in a particular class is compared with ResNet visual features of that class and all the other classes. This is repeated for all the GAN visual features of all the classes.

$$cos(\boldsymbol{X}, \boldsymbol{Y}) = \frac{\boldsymbol{X} \cdot \boldsymbol{Y}}{||\boldsymbol{X}|| \cdot ||\boldsymbol{Y}||} \tag{4.2}$$

4.2 Classification accuracy

In this section, I perform another set of experiments to emphasize the dissimilarity between the ResNet and GAN visual features. A common rule of thumb in any classification problems is that both the train set and test set should belong to the same distribution. If this is not the case, deteriorated performance on the test set could occur. The idea here is to build a classification problem. It is important to emphasize that we are performing classification rather than zero-shot

Table 4.2: Average inter and intra-class cosine similarity between GAN and ResNet visual features of unseen classes

Dataset	Cosine Similarity		Pearson's Correlation	
	Inter-class	Intra-class	Inter-class	Intra-class
CUB	0.42	0.41	0.01	0.01
FLO	0.49	0.38	0.08	0.03
AWA2	0.40	0.34	0.06	0.05

learning. Both the train and test dataset contain samples from all the classes. We divide the ResNet and GAN datasets into train and test sets. The ResNet dataset contains ResNet visual features of all the classes and the GAN dataset contains GAN visual features of all the classes. The datasets contain visual features of CUB dataset. We perform an 80:20 split for both the datasets. We train two classification models. Each classification model is trained on one train split. The accuracy of the trained classification is evaluated on both the test datasets. The results are shown in Table 4.3 .

Table 4.3: Testset Performance of the classification models. Network 1 is trained on ResNet visual features and Network 2 is trained on GAN visual features.

	ResNet Data	GAN Data
Network 1	91.31%	80.51%
Network 2	65.62%	99.96%

In table 4.3 , network 1 is trained on the ResNet visual features and network 2 is trained on GAN visual features. The table represents the performance of both the classification networks on the the test splits of both the datasets. The highlighted section in table 4.3 represents the accuracy of network 2 (trained on GAN visual features) on the ResNet visual features. The accuracy is 34% less than its performance on GAN visual visual features. This is because there is a domain gap between ResNet and GAN visual features. This result also explains that a network trained on GAN visual features might not perform well on ResNet visual features. As I described in Section 3.3, during the test time a classifier trained on GAN visual features is evaluated on ResNet Visual features. From this experiment, we can hypothesize that using ResNet visual features on a network trained on GAN visual features is the reason for lower zero-shot learning accuracy scores. At this point, it is important to consider some of the solutions for this problem.

4.3 t-SNE Visualization

Visualizing higher dimensional data can help to determine whether there are any patterns in the data. A possible way to plot higher dimensional data into 2D or 3D space is by using t-SNE [46]. t-SNE stands for t-Distributed Stochastic Neighbor Embedding. This technique is capable of preserving much of the local structure and global structure (such as presence of clusters) of the higher-dimensional data in the lower-dimensional space. Therefore, I chose t-SNE as our dimensionality technique; the goal here is to look at how the GAN visual features and ResNet visual features look like in lower dimensional space. Even though t-SNE tries to preserve the local and global structure of the data, it is inevitable that there would be some loss of data when compressing the data into a two-dimensional space from a 2048-dimensional space. Any observation in the lower dimensional space should be treated with caution. Fig 4.1, Fig 4.2 and Fig 4.3 show the t-SNE visualizations of GAN visual features and ResNet visual features on CUB, AWA and FLO datasets respectively.

In order to make the plots understandable, we only plotted the features for the first 10 classes in each of the three datasets. In these plots, we observed clusters of data points belonging to different classes. We can also see the cluster centroids of visual features belonging to different classes. In the plots, $resnet\_x$ represents the centroid of ResNet visual features of class x, whereas $fake\_x$ represents the generated or GAN visual features of the same class x. In an ideal scenario, we want both sets of visual features of class x to be as close as possible in any dimensional space. Highlighted in 4.1 is an instance where GAN visual features of one class are close to the ResNet visual features of another class. In this case, GAN visual features of class 8 are close to ResNet visual features of class 6. Scenarios like these would significantly reduce the classification accuracy. Analogous examples can be found in 4.2 and Fig 4.3. A general observation that can also be made from the three figures is that the GAN (or fake) visual features of a particular class are clustered at one place far enough from the GAN visual features of another class. However, the ResNet visual features of each class are spread all over the space and overlap with the ResNet visual features of other classes. This observation shows us that GAN, or fake, visual features are better suited for classification than the ResNet visual features.

4.4 Appending attributes

In Section 3.3, we observed that the GAN visual features were produced solely from attributes. The GAN was able to produce distinguishable attributes as seen in the tsne plots. Since the usage of attributes was able to produce distinguishable visual features, we wanted to include attribute information in the visual features of a ResNet in order to improve the performance of the Zero Shot Learning model. This can be done by finetuning a ResNet to predict both the attributes and visual features.

This experiment is designed such that the ResNet predicts visual features and attributes from an image and both these predictions are appended and fed to a classification layer to make a class prediction. The ResNet is trained on seen classes. The ResNet is trained to optimize a loss function that is a combination of mean squared error for the attribute prediction and classification loss for the class label prediction. Initially equal weight is given to both the losses.

We trained the ResNet model in this experiment on CUB dataset since the images were available publicly. During the training of the model we observed that the visual features predicted by the model were not improving. After careful investigation, we found that the network was not attempting to change the visual features; rather it was trying to improve its prediction of attributes. This is understandable because each class has a unique set of attributes which can be used to classify a image rather than using a combination of visual features and attributes. The training accuracy of the model was solely dependent upon attribute prediction because the model reduced the weights

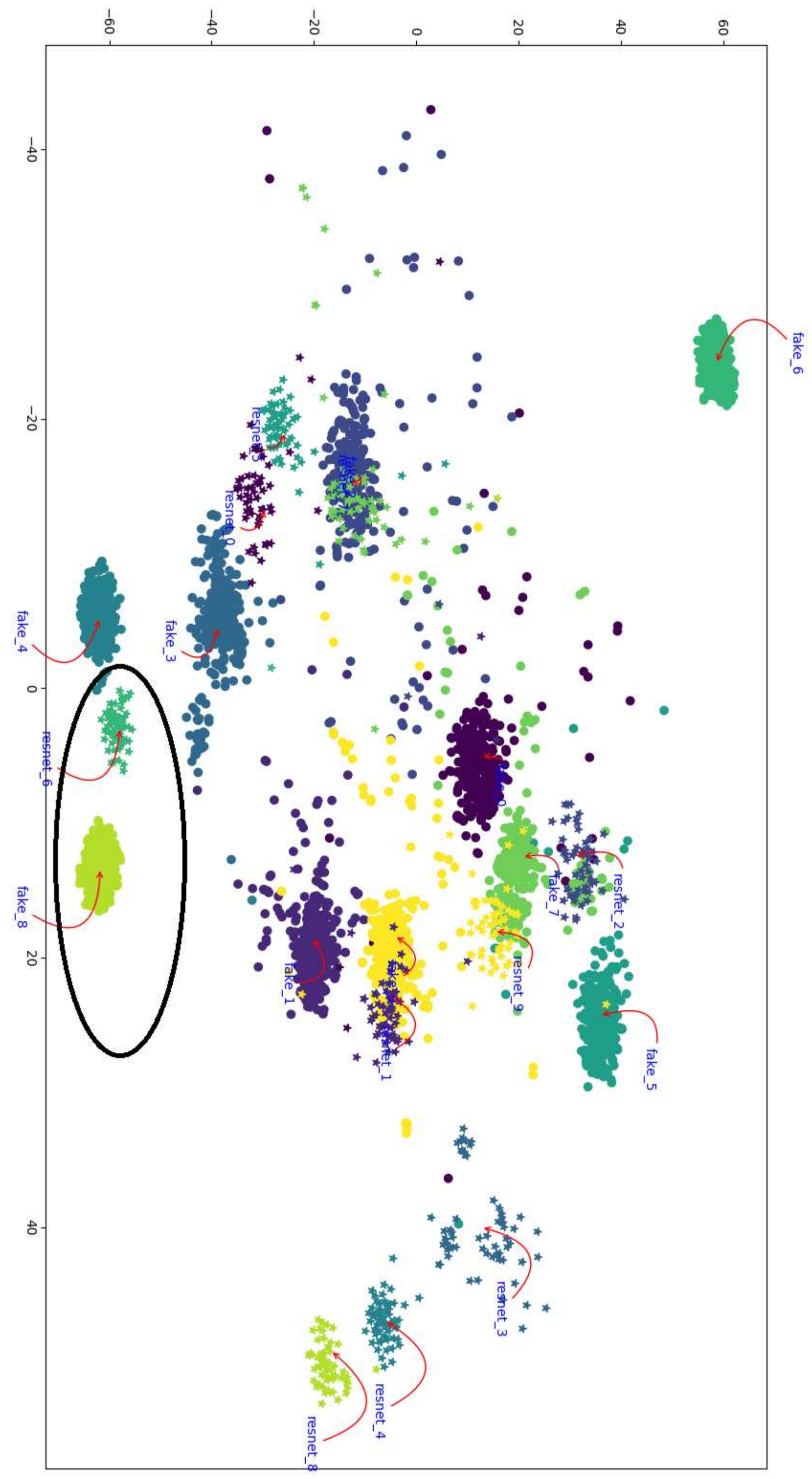

Figure 4.1: T - SNE visualizaion on CUB dataset

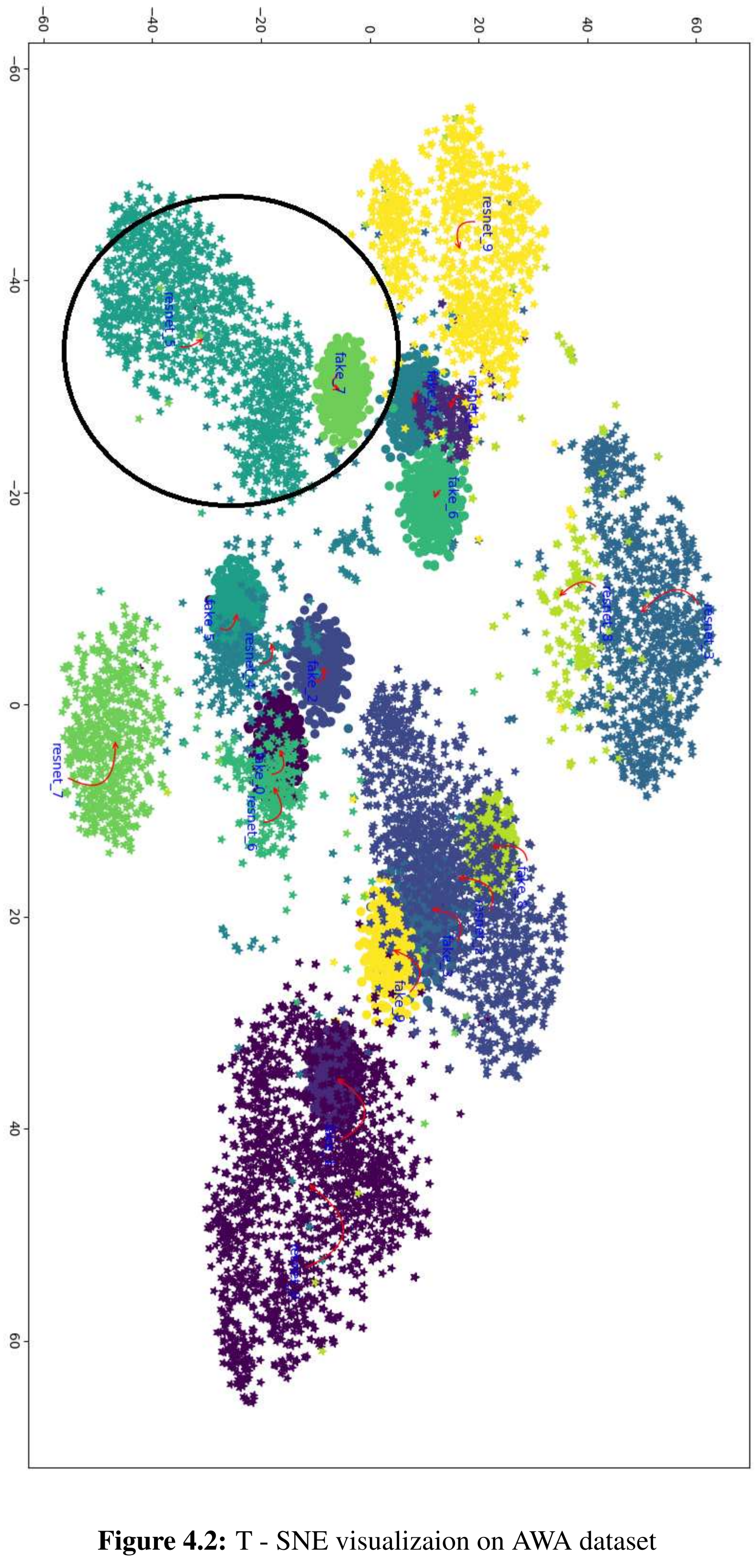

Figure 4.2: T - SNE visualizaion on AWA dataset

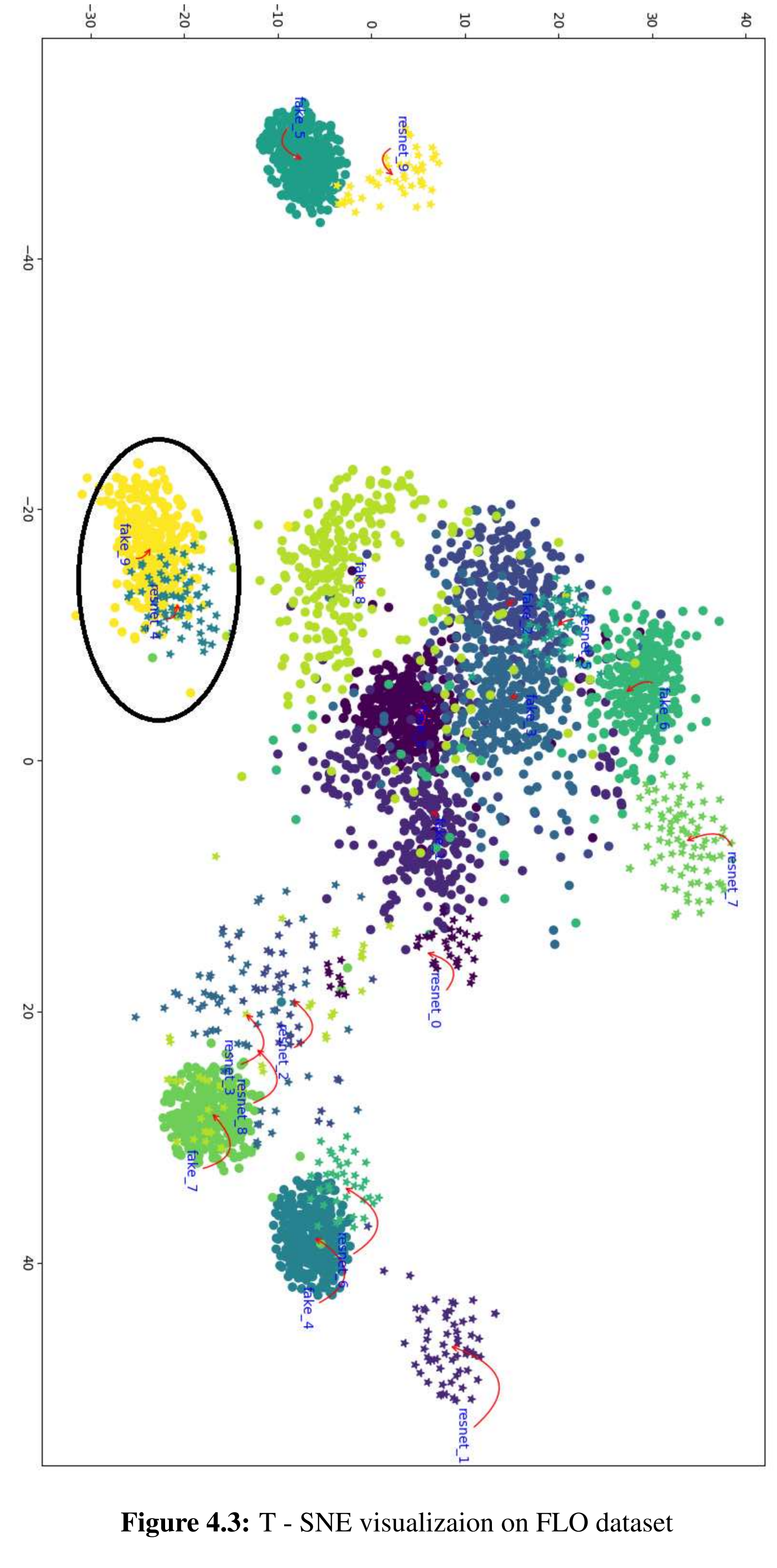

Figure 4.3: T - SNE visualizaion on FLO dataset

between the visual features and the classification layer. Thus, only the weights between the classification layer and attribute prediction layer were preserved. We failed to improve visual features and achieve our objective even by changing the weights in the loss function.

4.5 Autoencoder

Auto Encoders are neural networks that are used for the task of representation learning. In simple terms, auto encoders project the data onto a lower dimension space initially and try to reconstruct the data from the projected lower dimensional space. Figure 4.4 represents an autoencoder that projects the input data $\boldsymbol{x}$ onto lower dimensional space $\boldsymbol{a}$ and tries to reconstruct it as $\hat{\boldsymbol{x}}$. The goal of the autoencoder is to bridge the gap between $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}$. The hidden layer in the auto encoder is also called the "bottleneck."

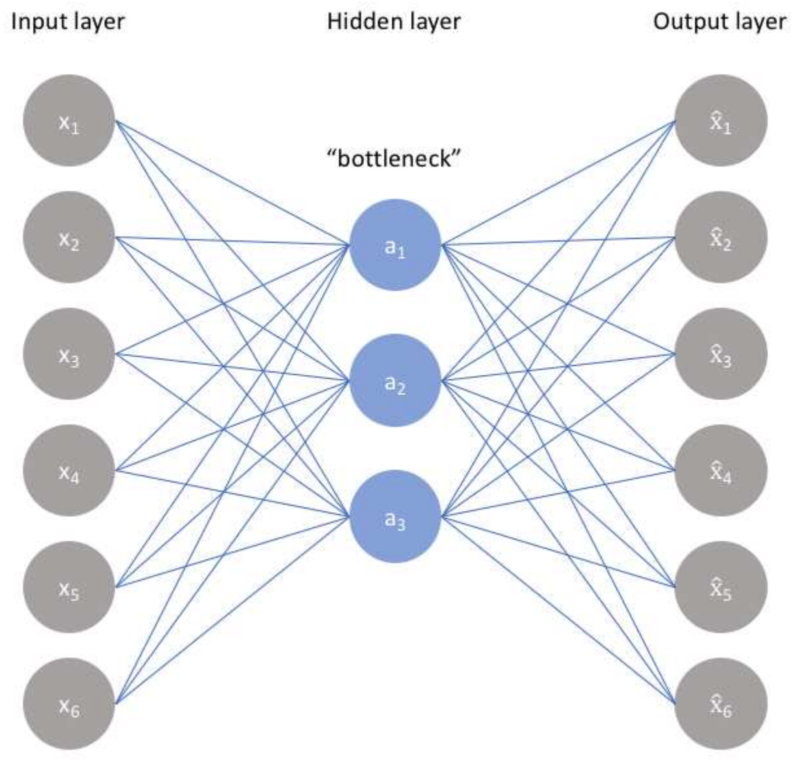

Figure 4.4: Auto Encoder. Image taken from [47]

In section 4.2, we explained why using ResNet visual features on a network trained on GAN visual features can be problematic. One possible solution to the problem is using a transformer that transforms ResNet visual features into the space of GAN visual features. The idea here is to transform the ResNet visual features into a space that has a high degree of similarity to GAN visual features. We use cosine similarity as a metric to measure the similitude between the transformed and GAN visual features. Autoencoder functioned as a transformer in this experiment. The autoencoder took ResNet visual features as input, compressed them in the bottleneck layer, and then projected them onto a space where the new transformed visual features are similar to GAN visual features. The role of the autoencoder in zero-shot learning comes into play during the test time once the GAN is trained. At the test time, the ResNet visual features were transformed using the autoencoder and were passed to the zero-shot classifier trained on GAN visual features for prediction.

Initially, the autoencoder was trained in zero-shot learning fashion. The autoencoder was trained to improve the cosine similarity between the ResNet and GAN visual features of seen classes. Ideally, once the autoencoder transforms seen class visual features with a high degree of similarity, it can transform the visual features of unseen classes. Cosine similarity is used as the loss function to train the autoencoder.

The autoencoder is trained and tested on AWE and CUB datasets separately. The performance of the autoencoder is evaluated on both generalized and conventional zero-shot learning settings. The autoencoder improved the similarity between the visual features of seen classes, but not of unseen classes. The autoencoder failed to improve the cosine similarity of unseen ResNet visual features. This affected the performance on zero-shot learning tasks. The autoencoder also performed poorly in generalized zero-shot learning settings. The autoencoder transformed the ResNet visual features of unseen classes into features that are similar to seen class GAN visual features rather than unseen class GAN visual features.

Table 4.4: Conventional Zero Shot Learning (CZSL) accuracy by using an autoencoder. We compare the Autoencoders results with authors CZSL results.

Dataset	Accuracy (CZSL) (in %)	
	Autoencoder	Author's
CUB	41.3	57.3
FLO	43.6	67.2
AWA2	49.3	68.2

4.6 Fine-tuning ResNet

In Section 3.3, we discussed how during the testing phase of zero-shot learning, a ResNet-101 was used to extract features from the test set images. In the section 4.2 above, we examined how ResNet visual features are not similar to GAN visual features on which the final classification model is trained on. In section 4.5, we tried to improve the cosine similarity between the ResNet visual features and GAN visual features using the autoencoder as a transformer. The autoencoder approach did not achieve satisfactory performance and failed to improve cosine similarity between both sets of visual features on the test set. Perhaps the autoencoder was not able to learn the complex relationship between both the feature spaces. In this section, we will try to exploit a ResNet model to improve the cosine similarity between both sets of visual features.

ResNet was introduced in [7], which showed that using residual connections or skip connections between the convolution layers helped to increase the depth of the network without significantly increasing the complexity of the network. The skip connections led to better features sharing and prevented the network from overfitting. The authors also proved ResNet's excellent generalization performance by demonstrating improved results on different recognition tasks like object detection and localization. The ResNet model has been rigorously trained on the ImageNet [9] dataset and achieved state of art accuracy on the dataset at that time. Since ResNet is such a complex network and is trained on a large dataset, it is well suited to produce visual features that are highly similar to GAN visual features upon fine tuning. The goal here is to use fine-tuned ResNet as visual feature extractor at test time.

Traditionally, ResNet-101 is used as a feature extractor for test set images in zero-shot learning. So, in this experiment, I chose to fine tune ResNet 101 to produce better visual features. Fine tuning the entire ResNet might not be a wise idea since all the zero-shot learning datasets have only a few thousands of images and ResNet 101 has 29.4M parameters [7]. We froze all the layers of the network except the last two convolution layers and the average pooling layer. We removed the classification head of the network since we want visual features rather than class labels. The ResNet we chose to fine tune was pretrained on ImageNet by the authors of PyTorch. The ResNet was fine-tuned separately on CUB and FLO dataset because they are the only datasets with publicly available images. During the training process, the ResNet was optimized to reduce the cosine embedding loss, which is similar to cosine similarity. The cosine embedding loss can be seen in the equation 4.3 below. The cosine embedding loss in 4.3 measures the similarity between samples x_1 and x_2, which would be visual features in our case. Because we were aiming to improve the cosine similarity, we always assigned the value of y to 1. It is important to note that as the cosine embedding loss decreases, the cosine similarity increases (up to 1). The cosine similarity is equal to 1 when the angle between the two vectors is 0°i.e when the vectors are parallel. The network is trained similarly to the Autoencoder in the above section, but it is trained on images instead of visual features. To recap, the ResNet101 is trained on seen class images of the dataset to improve the cosine similarity with the GAN visual features.

$$\text{CosineEmbeddingLoss(x,y)} = \begin{cases} 1 - cos(x_1, x_2), & \text{if } y = 1 \\ max(0, cos(x_1, x_2) - margin), & \text{if } y = -1 \end{cases} \quad (4.3)$$

Table 4.5: Average inter and intra-class cosine similarity between GAN and fine-tuned ResNet visual features of unseen classes. We can see that there is an increment in both inter-class and intra-class cosine similarity because of fine-tuning the ResNet

Dataset	Cosine Similarity	
	Inter-class	Intra-class
CUB	0.72	0.56
FLO	0.69	0.48

Table 4.6: Conventional Zero Shot Learning (CZSL) accuracy from finetuned ResNet visual features. We compare the new results with the authors' CZSL results. We observe a drop in accuracy even after fine-tuning the ResNet to match the GAN visual features.

Dataset	Accuracy (CZSL) (in %)	
	Finetuned ResNet	Author's
CUB	52.1	57.3
FLO	58.6	67.2

The ResNet 101 was initially trained on CUB dataset and its performance was tested on unseen classes. We initially analyzed the cosine similarity between the fine tuned ResNet visual features and CAN visual features of unseen classes. There was an increase in the cosine similarity, but it was not satisfactory enough. In table 4.5 we can notice that there is a boost in inter-class and intra-class cosine similarity of visual features after fine-tuning the ResNet-101. Usage of the new fine-tuned ResNet as a visual feature extractor at test time did not improve the zero-shot learning accuracy in both generalized and conventional zero-shot learning settings. Table 4.6 illustrates the performance of fine-tuned ResNet features versus the authors' reported results. We can see that there is a decline in the performance by using the new visual features from a fine-tuned ResNet. There were similar observations for the FLO dataset.In order to improve cosine similarity we tried to unfreeze and fine-tune more layers of the ResNet 101 which did not improve the results at all but rather led to overfitting on both the datasets.

It is important to understand why the ResNet failed to produce visual features that are similar to GAN visual features even after fine-tuning. We investigated the authors' implementation of F-CLSWGAN. It is important to recall that during the training process of F-CLSWGAN, a classification loss was used to optimize the GAN network to produce better visual features. The classification loss was obtained from a network that was trained on ResNet visual features of seen classes. More details of the training process can be found in Section 3.3. We found that the author trained F-CLSWGAN using the visual features extracted from original ResNet-101 (which is currently not available) published by the authors of ResNet in 2015. Based on this observation, there is one possible reason why our ResNet fine-tuning experiment failed. Primarily, the ResNet we fine-tuned in this experiment is totally different to the ResNet that was used by the authors to train F-CLSWGAN. Since the GAN visual features are not generated from the current ResNet visual features, the experiment failed. This conclusion raises two new questions. First, are all the ResNet 101 networks available in the deep learning frameworks the same? Second, can we repeat the experiment using a F-CLSWGAN re-trained with the ResNet available for us? These two questions are explored in the next section.

4.7 The Visual Feature Paradox

From the previous sections, we can understand that the primary reason why we are not able to achieve superior zero-shot learning accuracy is due to the fact that there is a serious mismatch between both the set of visual features (i.e. between ResNet and GAN visual features). Although ResNet visual features of the seen classes are used in the process of training of F-CLSWGAN, we still face the feature mismatch issue. In the last two sections, we tried to use different networks to bridge the gap between both the sets of visual features. In the last sections, we concluded that the main reason why we failed to train a network to produce visual features similar to the GAN visual features is the disparity between the version of ResNet 101 we used and the version the authors used to train F-CLSWGAN. In this section, we perform experiments to answer two main questions:

Table 4.7: Mean and Standard Deviation (SD) of different visual features

Dataset	Tensorflow		PyTorch		Author’s	
	Mean	SD	Mean	SD	Mean	SD
CUB	0.43	0.72	0.45	0.61	0.22	0.43
FLO	0.45	0.91	0.45	0.80	0.25	0.53

1) Is there any difference between the different ResNet-101 networks available on the market? and 2) Does the usage of the latest version of ResNet-101 affect the performance of F-CLSWGAN?

To determine whether there is any difference between the visual features of different versions of ResNet, we chose to compare the ResNet visual features provided by the authors with the two most widely used ResNet-101s. These widely used ResNet-101s are taken from deep learning frameworks PyTorch and Tensorflow. We compared the mean and standard deviation between the three visual features. We also assessed the similarities between the ResNet visual features used by the authors and the ResNet visual features extracted from latest version of ResNet-101 in PyTorch and Tensorflow using Cosine Similarity. Table 4.7 shows the mean and standard deviation of visual features used by the author to train the model and visual features taken from Tensorflow and PyTorch. We can see in the table that the mean and standard deviation the authors used is different to the ones extracted from Tensorflow and PyTorch. Table 4.8 depicts the cosine similarity of visual features from one framework to the other and with the visual features used by the authors to train the original model. We can see that the visual features from different deep learning frameworks have low cosine similarity with the authors’ visual features.

From the experiment described above, we can observe a clear distinction between the ResNet visual features used by the authors and the visual features from latest versions of ResNet. We also observe that the visual features produced by latest versions of Tensorflow and Pytorch are totally different. From this observation, we can assert that different versions of ResNet-101s from different deep learning frameworks significantly differ from each other. This explains the failure of

Table 4.8: Cosine Similarity between visual features (ResNet-101) from Tensorflow and PyTorch. The table represents the cosine similarity of visual features from one framework to the other and with the visual features used by the authors to train the original model.

Dataset	Tensorflow		PyTorch	
	Author's	PyTorch	Author's	Tensorflow
CUB	0.49	0.54	0.48	0.54
FLO	0.58	0.51	0.41	0.51

our fine-tuned ResNet to produce visual features similar to GAN visual features. Since the GAN that produced these GAN visual features is trained using the older version of ResNet, the latest version of ResNet-101 can not be trained to match these GAN visual features.

An easy solution to this problem is to retrain the GAN using the same process used by the authors with the visual features extracted from the latest versions of ResNet-101. We chose to initially retrain the GAN by extracting the visual features of the seen classes using the PyTorch implementation of ResNet-101. The reason we chose PyTorch over Tensorflow is because the authors of F-CLSWGAN published their work in PyTorch. We extracted the seen class ResNet visual features from the latest ResNet-101 in the same way authors extracted visual features from the older version of ResNet-101. We retrained the GAN on these new visual features on two datasets: CUB and FLO. These were the only datasets that had images publicly available. After looking at the results form the visual features of the PyTorch model, we also repeated the experiments using ResNet-101 from Tensorflow. The conventional Zero Shot Learning performance of the retrained GAN can be seen in Table 4.9 below. We used the models from Tensorflow 2.7 and PyTorch 1.10.2.

The results in Table 4.9 are quite surprising. The authors' results are highlighted in green. We see that there is a significant drop in the accuracy of the model once the older ResNet visual features are replaced with the latest ResNet visual features. When the model is retrained on visual features from ResNet-101 available in Tensorflow, the Conventional zero-shot learning accuracy dropped to 42% and 54% on CUB dataset and FLO dataset respectively. The accuracy further

Table 4.9: Zero Shot Learning accuracy on from different visual features

	CUB	FLO
Author's Visual Features	57.3%	67.2%
Tensorflow Visual Features	42.1%	53.9%
Pytorch Visual Features	4.3%	34.2%

declined when retrained on visual features taken from PyTorch 1.10.2. The model retrained on visual features taken from the PyTorch model achieved 34% accuracy on FLO dataset. We did not anticipate that the conventional zero-shot learning accuracy would drop to 4% from 57% on the CUB dataset while using visual features from latest PyTorch model. We acknowledged a certain degree of variability when we changed the visual features, but found the disparity in accuracy to be significant in this case. At this point, we want to re-iterate that we followed the exact visual feature extraction process as mentioned in the original paper. This raises serious concerns since we are unable to recreate the results using the latest versions of ResNet-101. There can also be potential questions on the validity of visual features used by the authors to train their networks in the first place.

Chapter 5

Future Directions

In Section 4.7, we discussed the difference between the weights of the same ResNet-101 in different deep learning frameworks. We also analyzed how this difference affects the performance of the zero-shot learning algorithm. The observations in Section 4.7 emphasize the need to build a framework that is agnostic to the changes in the weights of the same version of the model. A step in this direction would be standardizing the way images are used in zero-shot learning by building an input pipeline. The idea behind this approach is that variability in weights of the ResNet-101 should not significantly affect the performance of the zero-shot learning algorithm. In simple terms, the perception here is that the input pipeline should reduce the effect of change in the weights of a model on zero-shot learning algorithm. Another interesting avenue for further research would be to perform a study to determine how networks from different deep learning frameworks influence the performance of a zero-shot learning algorithm.

Bibliography

[1] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[2] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[5] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[6] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[8] Samuel F. Dodge and Lina J. Karam. A study and comparison of human and deep learning recognition performance under visual distortions. *CoRR*, abs/1705.02498, 2017.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[10] Chi Zhan, Dongyu She, Sicheng Zhao, Ming-Ming Cheng, and Jufeng Yang. Zero-shot emotion recognition via affective structural embedding. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1151–1160, 2019.

[11] Anjan Dutta and Zeynep Akata. Semantically tied paired cycle consistency for zero-shot sketch-based image retrieval. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5084–5093, 2019.

[12] Sounak Dey, Pau Riba, Anjan Dutta, Josep Lladós Lladós, and Yi-Zhe Song. Doodle to search: Practical zero-shot sketch-based image retrieval. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2174–2183, 2019.

[13] Ashish Mishra, Vinay Kumar Verma, M. Shiva Krishna Reddy, Arulkumar S., Piyush Rai, and Anurag Mittal. A generative approach to zero-shot and few-shot action recognition. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 372–380, 2018.

[14] Jie Qin, Li Liu, Ling Shao, Fumin Shen, Bingbing Ni, Jiaxin Chen, and Yunhong Wang. Zero-shot action recognition with error-correcting output codes. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1042–1051, 2017.

[15] Lajanugen Logeswaran, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, Jacob Devlin, and Honglak Lee. Zero-shot entity linking by reading entity descriptions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3449–3460, Florence, Italy, July 2019. Association for Computational Linguistics.

[16] Mikel Artetxe and Holger Schwenk. Massively Multilingual Sentence Embeddings for Zero-Shot Cross-Lingual Transfer and Beyond. *Transactions of the Association for Computational Linguistics*, 7:597–610, 09 2019.

[17] Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and

Jeffrey Dean. Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. *Transactions of the Association for Computational Linguistics*, 5:339–351, 10 2017.

[18] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[19] Yongqin Xian, Bernt Schiele, and Zeynep Akata. Zero-shot learning - the good, the bad and the ugly. *CoRR*, abs/1703.04394, 2017.

[20] Yongqin Xian, Tobias Lorenz, Bernt Schiele, and Zeynep Akata. Feature generating networks for zero-shot learning. *CoRR*, abs/1712.00981, 2017.

[21] Zeynep Akata, Florent Perronnin, Zaïd Harchaoui, and Cordelia Schmid. Label-embedding for image classification. *CoRR*, abs/1503.08677, 2015.

[22] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Marc' Aurelio Ranzato, and Tomas Mikolov. Devise: A deep visual-semantic embedding model. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[23] Elyor Kodirov, Tao Xiang, and Shaogang Gong. Semantic autoencoder for zero-shot learning. *CoRR*, abs/1704.08345, 2017.

[24] Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[25] Yongqin Xian, Zeynep Akata, Gaurav Sharma, Quynh N. Nguyen, Matthias Hein, and Bernt Schiele. Latent embeddings for zero-shot classification. *CoRR*, abs/1603.08895, 2016.

[26] Ruizhi Qiao, Lingqiao Liu, Chunhua Shen, and Anton van den Hengel. Less is more: zero-shot learning from online textual documents with noise suppression. *CoRR*, abs/1604.01146, 2016.

[27] Ziming Zhang and Venkatesh Saligrama. Zero-shot learning via semantic similarity embedding. *CoRR*, abs/1509.04767, 2015.

[28] Mohammad Norouzi, Tomas Mikolov, Samy Bengio, Yoram Singer, Jonathon Shlens, Andrea Frome, G.s Corrado, and Jeffrey Dean. Zero-shot learning by convex combination of semantic embeddings. 12 2013.

[29] Soravit Changpinyo, Wei-Lun Chao, Boqing Gong, and Fei Sha. Synthesized classifiers for zero-shot learning. *CoRR*, abs/1603.00550, 2016.

[30] Maxime Bucher, Stéphane Herbin, and Frédéric Jurie. Improving semantic embedding consistency by metric learning for zero-shot classification. *CoRR*, abs/1607.08085, 2016.

[31] Li Zhang, Tao Xiang, and Shaogang Gong. Learning a deep embedding model for zero-shot learning. *CoRR*, abs/1611.05088, 2016.

[32] Yanan Li, Donghui Wang, Huanhang Hu, Yuetan Lin, and Yueting Zhuang. Zero-shot recognition using dual visual-semantic mapping paths. *CoRR*, abs/1703.05002, 2017.

[33] Nour Karessli, Zeynep Akata, Andreas Bulling, and Bernt Schiele. Gaze embeddings for zero-shot image classification. *CoRR*, abs/1611.09309, 2016.

[34] Meng Ye and Yuhong Guo. Zero-shot classification with discriminative semantic representation learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5103–5111, 2017.

[35] Yanwei Fu, Timothy M. Hospedales, Tao Xiang, and Shaogang Gong. Transductive multi-view zero-shot learning. *CoRR*, abs/1501.04560, 2015.

[36] Vinay Kumar Verma and Piyush Rai. A simple exponential family framework for zero-shot learning. *CoRR*, abs/1707.08040, 2017.

[37] Wenlin Wang, Yunchen Pu, Vinay Kumar Verma, Kai Fan, Yizhe Zhang, Changyou Chen, Piyush Rai, and Lawrence Carin. Zero-shot learning via class-conditioned deep generative models. *CoRR*, abs/1711.05820, 2017.

[38] Bin Tong, Martin Klinkigt, Junwen Chen, Xiankun Cui, Quan Kong, Tomokazu Murakami, and Yoshiyuki Kobayashi. Adversarial zero-shot learning with semantic augmentation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[39] Yizhe Zhu, Mohamed Elhoseiny, Bingchen Liu, and Ahmed M. Elgammal. Imagine it for me: Generative adversarial approach for zero-shot learning from noisy texts. *CoRR*, abs/1712.01381, 2017.

[40] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[41] Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3485–3492, 2010.

[42] Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 951–958, 2009.

[43] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings*

of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, page 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[44] Jason Brownlee. A gentle introduction to generative adversarial networks (gans), 2019. [Online; accessed March 27, 2019].

[45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[46] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[47] JEREMY JORDAN. Introduction to autoencoders., 2018. [Online; accessed March 25, 2019].