Deep Convolutional Generative Adversarial Networks

Isabella Gomez
Electrical and Computer Engineering
A15305555

Nicholas Muñoz
Electrical and Computer Engineering
A16117845

Abstract

Generative Adversarial Networks (GANs) have emerged as a prominent area of research which continues to have remarkable advancements in various applications. Among the GAN variants, Deep Convolutional GANs (DCGANs) have emerged as a powerful architecture, leveraging deep convolutional neural networks (CNNs) to enhance image generation capabilities. In this work, we explore the potential of DCGANs in improving image generation performance, along with their applicability to feature learning and supervised tasks such as image classification. By conducting experiments on multiple datasets, we demonstrate the effectiveness of DCGANs in generating high-quality images, even with limited data and reduced image sizes. Moreover, we show that a GAN discriminator can be used as a feature extractor for which we employ various methods, and achieve comparable results. Lastly, we show that while a GAN discriminator can be utilized as a feature extractor, there must be further improvements done in order to achieve satisfactory results. Our work highlights the strengths of DCGANs in image synthesis and feature learning while also shedding light on areas that require further attention to maximize their potential.

1 Introduction

1.1 GANs

GANs consist of two neural networks, the generator and the discriminator, engaged in a competitive training process. The generator synthesizes new data samples, while the discriminator learns to distinguish between real and fake samples. Through adversarial training, both networks improve their performance over time. The primary purpose of GANs is image generation and manipulation, areas where they have demonstrated remarkable capabilities. GANs have evolved significantly since their introduction in 2014 by Goodfellow et. al [1]. They continue to be an active area of research and have made remarkable advancements in various applications.

1.2 DCGANs

DCGANs are a specific type of GAN architecture that utilizes deep CNNs for image generation tasks. They were introduced in 2015 by Radford et al. [3] and have since become a foundational model in the GAN literature. DCGANs incorporate key architectural changes to enhance the performance of traditional GANs.

The key components of the DCGAN architecture as stated in [3] include:

- Replace pooling layers with strided convolutions in the discriminator and fractional-strided or transposed convolutions in the generator.
- Use batchnorm after the convolutional layers in the generator and the discriminator.
- Remove fully connected hidden layers.

- The generator uses ReLU activation function in all layers except for the output, adn the output uses the Tanh activation function.
- The discriminator uses LeakyReLU activation for all layers

DCGANs have gained wide adoption and have been extended for various applications, including image synthesis, image-to-image translation, and feature learning. While more complex architectures have been proposed in recent years, DCGANs continue to serve as a fundamental reference point in the field due to their straightforward yet effective design principles. They have laid the groundwork for subsequent advancements and inspired further research in the realm of GANs.

In this paper we design and train a DCGAN generator and discriminator which produce realistic images corresponding to the datasets we use. After this, we use the trained models and the discriminator in order to successfully train a classification L2-SVM algorithm. We show that this combined DCGAN and L2-SVM approach performs with higher accuracy than other established classification algorithms.

2 Related Work

2.1 Generative Adversarial Nets

Generative Adversarial Networks (GANs) have become a popular study topic since it was first proposed by Goodfellow et. al [1]. These consist of two neural networks, a generator and a discriminator. Each of these are trained simultaneously in a kind of game where one wins when the other makes a mistake. The generator attempts to formulate images based on a given dataset and wins when the discriminator is unable to distinguish between the generated images and the dataset. The model that makes the mistake, or loses, is updated while the other remains the same. While this work is innovative, it is not used for any supervised tasks and can produce noisy unintelligible images. Our work uses the GAN as a framework to create Deep Convolutional GANs and expand the use in supervised tasks, such as image classification.

2.2 Semi-Supervised Learning with GANs

There are various methods of semi-supervised learning using GANs. One of these is to estimate the tangent space by training the GAN generator to learn the data manifold through generating realistic samples that resemble real data. Kumar et al. [2] proposed a semi-supervised learning approach that enhances encoder training and improves the similarity between generated images and the input dataset. They utilized an Augmented-BiGAN, where the encoder and generator are trained simultaneously. Moreover, they observed that incorporating feature matching, which matches the feature representations of real and generated samples, leads to improved accuracy. Their work demonstrates that the GAN generator estimates the tangent space of the data manifold, resulting in increased accuracy for semi-supervised tasks through the injection of invariances into the classifier/discriminator. In our work, we focus on an unsupervised training method due to the abundance of unlabeled data instead of a semi-supervised approach.

2.3 Improved Techniques for Training GANs

In Salimans et al. [4] they offer various techniques to improve the semi-supervised learning performance and to improve the samples generated. This is done by focusing on the convergence of the GAN in training. In this work they use feature matching, minibatch discrimination, historical averaging, and more to improve the quality of the images produced. In [4] they also focus on providing an evaluation of their images using humans for a visual Turing test. This work provides great insight into the various methods and techniques that can be used to generate realistic images. In our work, we leverage the trained discriminator for supervised tasks, prioritizing the properties that influence the generated images rather than their realism from a human perspective.

3 Method

3.1 Data Preprocessing

In this work we use 3 datasets (Pet, Car, and Faces) described in § 4.1. Each of the datasets needed to be resized to the size of our GAN. Given the resources available to us we decided to resize our images to 64x64 pixels. This allowed us to run our model for extended periods of times and perform more passes over the dataset for training. Normalizing the images was also performed in order to help lower the computational cost of training the images. The same procedures were done on the COCO dataset since it is used as input to the GAN's discriminator for extracting features and classification described in more detail in § 3.5.

3.2 GAN architecture

3.2.1 Generator

The generator architecture consists of several layers that aim to transform a noise vector into a realistic image. The process begins with an input noise vector which is continuously upsampled. The generator architecture has four transposed convolutional layers that are followed by batch normalization and use the ReLU activation function. These layers are followed by a last transposed convolutional layer which is passed through a hyperbolic tangent (Tanh) activation function and produces the final output. The final layer does not use batch normalization and outputs images that are the same size as the training images.

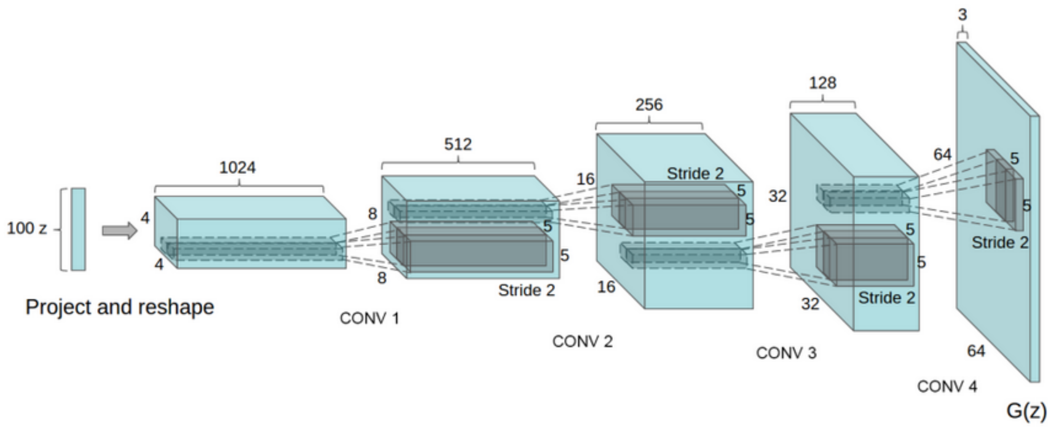

Figure 1: Generator architecture diagram.

3.2.2 Discriminator

The discriminator consists of 4 convolutional layers, each followed by a leaky ReLU activation function to introduce non-linearity. The process begins with an input that corresponds to the number of color channels of the input images. The discriminator has 4 convolutional layers each with batch normalization and which are subsequently passed through a LeakyReLU activation function. After these, the discriminator has a final convolutional layer which uses a Sigmoid activation function to produce the final output. The final layer does not use batch normalization.

3.3 Hyperparameter Tuning

Our architecture hyperparameters were based on [3] and set to the following values:

- Learning rate = 0.0002
- Beta1 hyperparameter for Adam optimizers = 0.5
- Beta2 hyperparameter for Adam optimizers = 0.999
- Size of z latent vector = 100

- Number of channels in the training images = 3
- Batch size used in training = 128
- Epochs = 100
- Feature map depth for generator and discriminator = 64

The weights for the models were randomly initialized from a Normal distribution with mean=0 and standard deviation = 0.02. The optimizers that were chosen for this task was the Adam optimizer with beta values as previously mentioned. Finally, the loss function was Binary Cross Entropy loss (BCELoss).

Discriminator D	**Generator** G
Input 3x64x64 image	Input $\in \mathbb{R}^{100}$
4 x 4 conv. 64 lRELU. stride 2	4 x 4 convTrans. 512 batchnorm RELU. stride 2
4 x 4 conv. 128 batchnorm lRELU. stride 2	4 x 4 convTrans. 256 batchnorm RELU. stride 2
4 x 4 conv. 256 batchnorm lRELU. stride 2	4 x 4 convTrans. 128 batchnorm RELU. stride 2
4 x 4 conv. 512 batchnorm lRELU. stride 2	4 x 4 convTrans. 64 batchnorm RELU. stride 2
4 x 4 conv. 1 stride 1 sigmoid	4 x 4 convTrans. 3 TANH. stride 2

Table 1: The Discriminator and Generator used for images

3.4 DCGAN Training

The training loop for the DCGAN involved training both the generator and discriminator at the same time. Firstly, the discriminator was trained on real images taken from our datasets, and then was trained on false images that were taken from the generator. This is done in order to maximize the probability of classifying an input correctly as real or fake. After training the discriminator on both batches, we call the optimizer and update the discriminator.

The next step in the training loop is training the generator. This involves passing the generator output through the discriminator for classification of either real or fake, computing the gradient in a backward pass and finally updating the generator with the optimizer.

During training we collect the images output by the generator, the generator and the discriminator loss, as well as the average discriminator outputs from the real and fake batches. The DCGAN was trained on the Face, Car, and Pet datasets and at the end of the training loop these models are saved to be reloaded for later use.

3.5 Feature Extraction Methods

Feature extraction is a process that utilizes dimensionality reduction to identify the most representative features from a given dataset while effectively eliminating noise and redundant information. Feature extraction was used by [3] where they use their GAN as a feature extractor on a supervised dataset in order to evaluate linear models that used these features for classification.

In this work, we apply the same methodology with our GAN for the different datasets described in § 4.1. We investigate two methods of feature extraction. The first is following the methodology described in [3] where they extracted the feature maps following each convolutional layer and use aggregation operators to prepare it for the linear models for classification. The second method utilizes the last layer of the discriminator network for feature extraction instead of the aggregation of feature maps from all convolutional layers.

These two methods' performances were then compared to show whether there is significant difference between using the earlier layers in our model for feature extraction and using the later layers. As seen in Yosinski et al. [6] there is a distinction in the learned outputs between the earlier and later layers. The later layers tend to be more dataset-specific and, consequently, hold potential in defining discriminative features for classification purposes. This shows the relevance of exploring the characteristics and contributions of different layers within the model architecture.

We will be using 3 different instances of GAN networks where each is trained on a single single dataset described in § 4.1 in order to compare performance. Both methods will be used for fitting linear models in order to evaluate the performance of the pre-trained GAN.

Feature Extraction Procedure 1 Following the feature extraction method described in [3]. We took our discriminator network from our GAN and altered its architecture to allow the saving of feature maps as it passes batches through its forward pass. Prior to the feature map collection the saved weights for each GAN instance (corresponding GAN per dataset) are uploaded so that the discriminator network has the pre-trained weights. Each convolutional layer results in a feature map and each feature map is saved for processing. Once the feature maps from all the layers are recorded they are then passed to a pooling layer that produces a 4x4 grid spatial grid for each corresponding feature map. Each of these spatial grids are then flattened and concatenated to form a 1-dimensional feature vector for each corresponding image. This procedure is used on the COCO dataset described in § 4.1 to test performance through classification.

Feature Extraction Procedure 2 The second approach uses the discriminator as a feature extractor by using its architecture as a base to define a new FeatureExtraction class. This class contains 4 convolutional layers with batch normalization and ReLU activation function and differs from the discriminator by excluding the last convolutional layer from the FeatureExtraction class. The procedure for feature extraction begins by loading the weights that were saved after training the DCGAN on the corresponding Pet, Car, and Face datasets and instantiating a discriminator instance with them. Then, we create 3 instances of the FeatureExtraction class using each discriminator instance, and transfer the weights from the loaded discriminators to the feature extractors. Finally we iterate over the COCO val2017 dataset and pass the images through the feature extractors to obtain the intermediate feature representations. In total we retrieved 3 feature maps from the COCO images, one per training dataset which can then be used to train separate classifiers on the COCO val2017 dataset for classification.

3.6 Classification Using Extracted Features

After we obtain the extracted features from the COCO images we use the L2-SVM algorithm to implement image classification on them. In this case we use the scikit-learn library to split the data into test and train data, initialize and train the L2-SVM model, as well as make predictions and evaluate the model. The accuracy is printed as the final output. The hyperparameter values chosen for this implementation are:

- kernel='rbf'
- C=.0001

This process was done 3 times, one for each of the Pet, Face, and Car extracted feature maps and each of their accuracies was stored and compared.

3.7 Comparative Analysis

Finally, we also implemented a KNN algorithm to train and test the COCO dataset. This was done in order to compare the output performance of our DCGAN + L2-SVM method with other conventional and established methods of image classification.

4 Experiments

4.1 Datasets

4.1.1 Oxford-IIIT Pet Dataset

The Oxford-IIIT Pet Dataset consists of RGB images of dogs and cats of various breeds. It contains roughly 200 images of each breed of animal with a total of 37 different breeds, therefore, the dataset contains around 7,400 images. The images are all different sizes and resolutions and are in JPG format. The animals in the dataset are in different lighting conditions and in different locations and positions. The images range from full body, partial body, and facial images. The dataset was cleaned by making the images square of sizes 64x64 pixels by resizing and then were normalized. The dataset was shrunk due to corrupted image files and to account for even batches of 128 images per batch. With these changes, the final dataset used became 7296 images.

4.1.2 Flickr Faces HQ Dataset

The Flickr Faces HQ Dataset consists of 70,000 PNG images in 128x128 resolution of faces of people of different races, ages, and sexes. The images have a great variety of backgrounds and include images of people with different accessories such as glasses, hats, makeup, etc. These images are also resized to be 64x64 in dimension. The dataset was also reduced to 10,880 images for the sake of a more efficient training time and to match the batch size of 128.

4.1.3 Comprehensive Car Dataset

The Comprehensive Car Dataset (Yang et. al [5]) is a dataset of surveillance style images of parked cars from garages and outdoor settings. The cars are in multiple positions, facing in different directions, and are in different lighting conditions. Of the total 136,726 images of complete cars we kept 16,000 for the sake of training time and to keep an appropriate batch size of 128 images per batch. The images are in JPG format and were also reduced to be squares of 64x64 pixels.

4.1.4 COCO 2017 Validation Dataset

The COCO dataset is a labeled common objects dataset that contains a variety of images of many different subjects and classes. The 2017 Validation subset of COCO holds 5,000 images of people, animals, objects, etc. both in indoor and outdoor settings. The images are in JPG format and are all different sizes, have different lighting conditions, and picture items in different positions in the image. These images were again reduced in size to be 64x64 for the sake of training time and a total of 4992 images were kept in order to have the appropriate batch size of 128 images per batch.

4.2 Results

4.2.1 Generated Images

The images generated by our trained DCGAN for the Pet dataset are seen in Figure 2. These images are not entirely defined nor discernible. Compared to the other generated images, the Pet dataset produces the worst quality false images. This makes sense because this is the smallest dataset in the bunch. While the animal in the image is not completely defined, it is easily noticeable that many of the images are either outdoors, in grass, or indoors. In some images, but not all, it is also possible to tell that the figure in the image is an animal like a dog or cat.

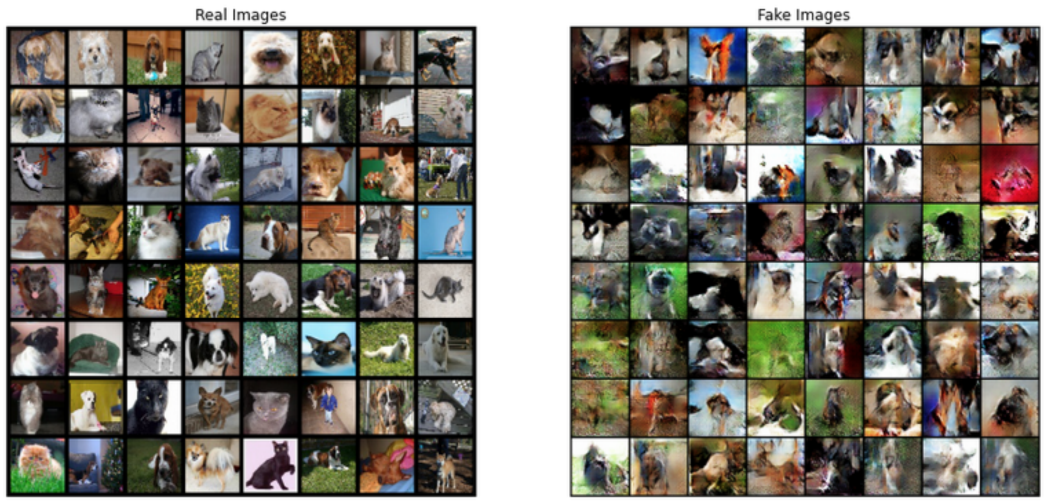

Figure 2: Real and false/generated images corresponding to the Pet dataset.

The images generated by the Face dataset (as seen in Figure 3), on the other hand, look well defined and it is possible to easily tell they are human faces in the pictures. The faces produced also all look different, as they are all in different location, lighting, and portray humans of diverse background.

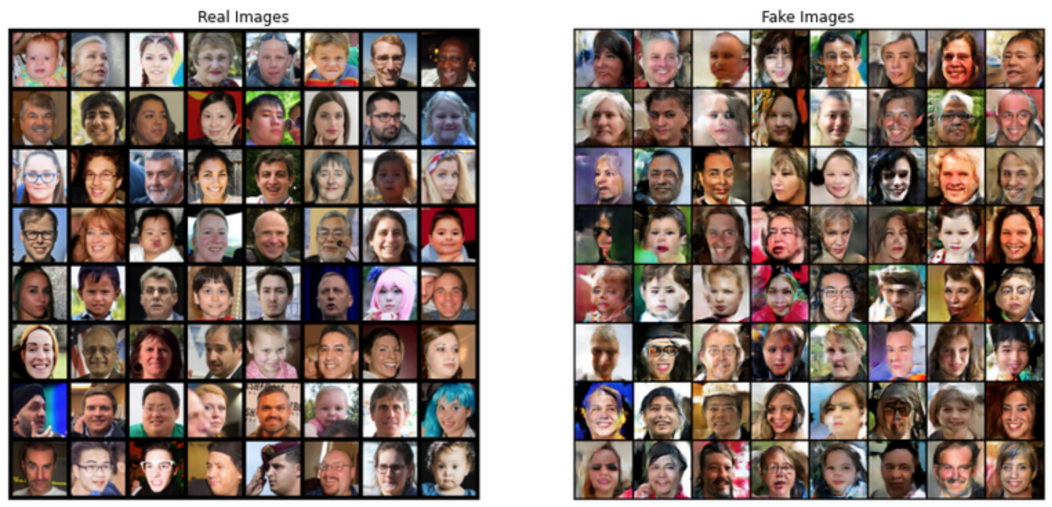

Figure 3: Real and false/generated images corresponding to the Face dataset.

Similarly to the Face dataset, the images generated by the Comprehensive Car dataset (as seen in Figure 4) are crisp and for the most part well defined. It is immediately discernible that the images represent cars and these are also in different locations. Some images portray the vehicles in an indoor garage and some clearly outdoors. Overall, the car generated images and the face generated images are easily comprehensible, while the pet dataset is less so. Given that the car and face datasets are the largest, we can conclude that the number of images that are used in training the DCGANs have a significant impact on the output of the generator.

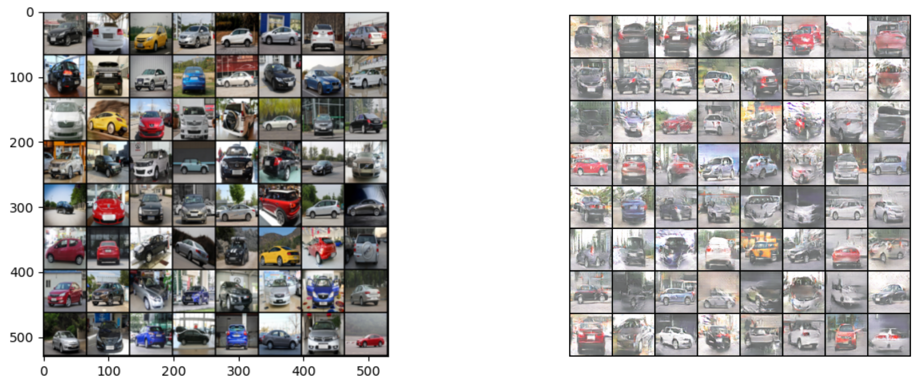

Figure 4: Real and false/generated images corresponding to the Car Dataset

4.2.2 Image Classification Comparison

The image classification table shows the different error rates for different combinations of algorithms and methods. The lowest error rate on the table is for the DC-GAN + L2-SVM algorithm for each of the datasets. These error rates are lower than the other more established algorithms such as KNN and a generic CNN architecture for image classification. This shows that our method is comparable to these other methods for image classification.

Overall, the results for the error rate for all our image classification algorithms is quite high. The first reason for this is that the dataset we use for image classification is the COCO 2017 Validation dataset.

This dataset consists of only 5,000 images which were then split into training and validation sets. Therefore, the size of our training set resulted to be very small. Furthermore, the COCO dataset has 92 different 'stuff' categories that our algorithms were attempting to classify. This contrasts highly with [3] given that in their tests they use the Cifar-10 dataset, which only hosts 10 different categories.

Model	Error Rate
KNN	83.2%
L2-LinearSVM	97.4%
L2-SVM	82.18%
CNN	80.7%
DCGAN(Pet) + L2-LinearSVM	97.4%
DCGAN(Pet) + L2-SVM	82.18%
DCGAN(Face) + L2-LinearSVM	97.4%
DCGAN(Face) + L2-SVM	82.18%
DCGAN(Car) + L2-LinearSVM	97.4%
DCGAN(Car) + L2-SVM	82.18%

Table 2: COCO classification results using DCGAN as feature extractor

4.2.3 Feature Extraction Method Comparison

The two different feature extraction methods ended up producing the same results. Therefore, we can conclude that both methods for feature extraction are valid in this regard.

5 Discussion and Conclusions

In this work we looked to implement multiple instances of GAN's by training each based on a single dataset out of the 3 defined in § 4.1. From tuning the hyperparameters of these networks we achieved recognizable results that were considered enough for our experiments. We were interested in the work described in [3] where they used the GAN's as feature extractors on supervised datasets in order to evaluate the performance of the GAN's. In the paper, they evaluated the performance of their GAN by pre-training it on Imagenet1k and then used it as a feature extractor to evaluate the performance of it on CIFAR10/100. From expanding on this work and following their same procedure on the 3 datasets defined in § 4.1, we received the following results in Table 3 from evaluating on the COCO dataset. We applied two forms of L2 regularized SVM's and found that we received the same results between the different pre-trained models. This confirms that given that different datasets were used for training the model will still produce the same results for evaluation because they were based on the same GAN architecture. We compared two types of feature extraction by following the papers methodology by pulling feature maps from each convolutional layer to only the last feature map. From both methods the performance of our linear models with the extracted features didn't perform well. By using the same linear models, but with the full set of features defined from COCO dataset for classification to determine baseline performance metrics to compare our linear models using extracted features. The results can be seen in Table 2. From comparing our models using the extracted features for classification we can see that we achieve comparable results except comparing to the CNN architecture which achieve the lowest error rate. From this we can see that performing classification on the COCO dataset is a challenge classificaiton task compared to CIFAR10/100.

Overall from this work we can see that pre-training a model on different datasets will get comparable results when evaluating the GAN's through feature extraction on supervised datasets. Comparing the results to other models we can see that the performance may not be as great as others on tougher datasets like the COCO dataset. We were also able to see from our results that applying feature extraction by taking the last layer or aggregating all the layers resulted in similar results. In future works we feel that given our resources we explored smaller a smaller GAN's for these datasets, but feel that for larger GAN's these results may vary in terms of performance between datasets.

Model	Error Rate
Feature Extraction procedure 1:	
DCGAN(Pet) + L2-SVM	82.18%
DCGAN(Face) + L2-SVM	82.18%
DCGAN(Car) + L2-SVM	82.18%
Feature Extraction procedure 2:	
DCGAN(Pet) + L2-SVM	82.18%
DCGAN(Face) + L2-SVM	82.18%
DCGAN(Car) + L2-SVM	82.18%

Table 3: COCO classification results using the two different feature extractor methods

6 Team member contribution

Our team members equally contributed to this project. Project code can be accessed at `https://drive.google.com/drive/folders/1eJf88F9V-8dy4XT147zoyzODj8PTIMaY?usp=sharing`.

7 Supplementary Material

Here we list a few other iterations of the generated images from the Pet and Face dataset in Figure 6. We also show the loss produced from the generator and discriminator while training both datasets (Pet and Face) during 100 epochs of 128 image batches in Figure 5.

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. `https://papers.nips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf`, 2014.

[2] Abhishek Kumar, Prasanna Sattigeri, and P. Thomas Fletcher. Semi-supervised learning with gans: Manifold invariance with improved inference. `https://proceedings.neurips.cc/paper_files/paper/2017/file/d3d80b656929a5bc0fa34381bf42fbdd-Paper.pdf`, 2017.

[3] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. `https://arxiv.org/pdf/1511.06434.pdf`, 2016.

[4] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. `https://proceedings.neurips.cc/paper_files/paper/2016/file/8a3363abe792db2d8761d6403605aeb7-Paper.pdf`, 2016.

[5] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. A large-scale car dataset for fine-grained categorization and verification. `https://arxiv.org/pdf/1506.08959.pdf`, 2015.

[6] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? `https://arxiv.org/pdf/1411.1792.pdf`, 2014.

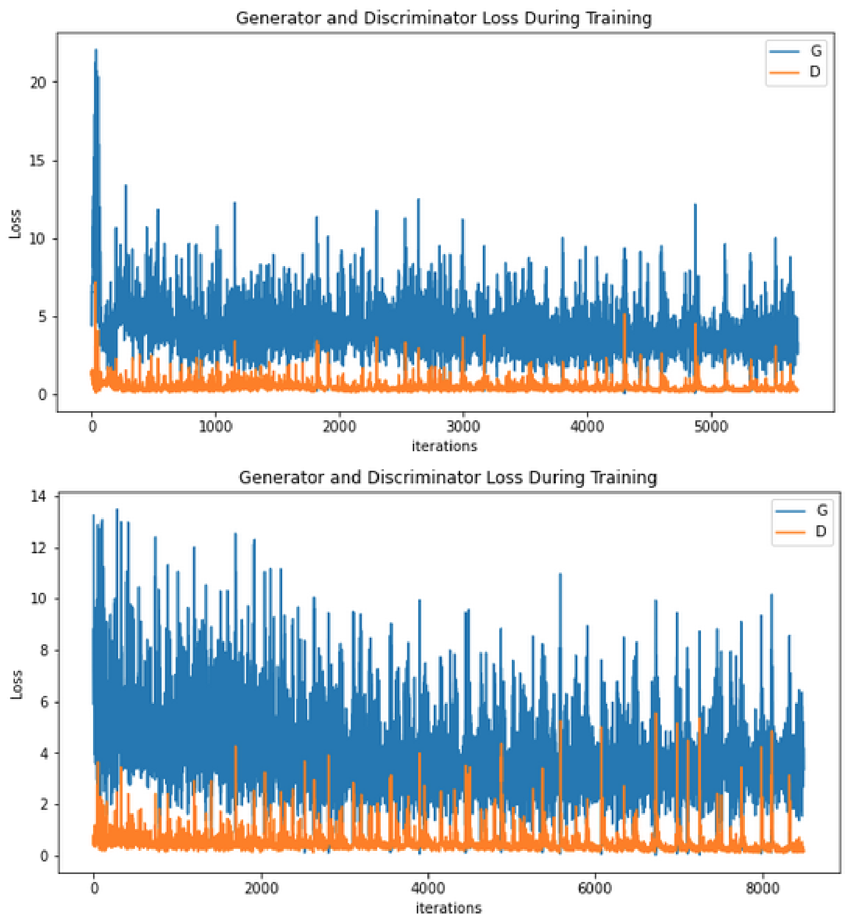

Figure 5: Loss of Generator and Discriminator corresponding to Pet and Face dataset training respectively.

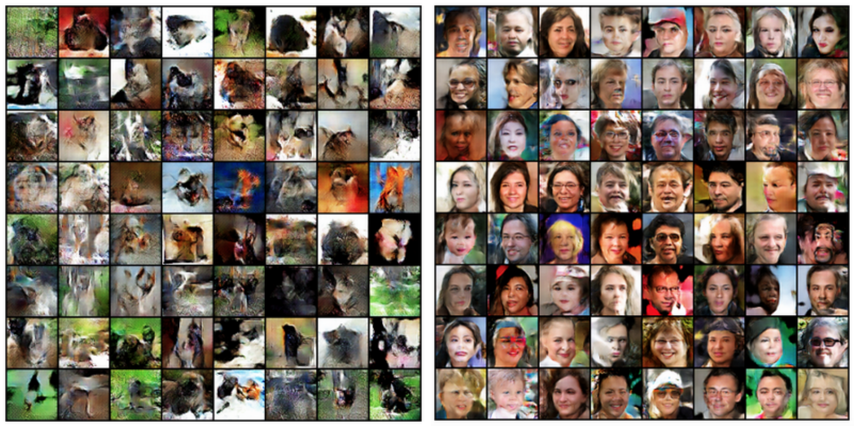

Figure 6: More false/generated images corresponding to the Face and Pet dataset.