Improving Image Classification with Deep Convolutional GAN Data Augmentation

Machine Vision – M1- International Track in Electrical Engineering

Carine Farhat
Department of Electrical Engineering,
Université d'Evry,
Université Paris-Saclay
Evry, France
carine_farhat@outlook.com

Abstract — Data scarcity and data incompatibility are major problems of artificial neural networks applications. They cause the networks to overfit the train data and prevent them from achieving good results in testing and real-life applications. Generative Adversarial Networks (GAN) have been gaining attention in data augmentation over the past few years. In this paper we implement a GAN to augment datasets of cows and horses and evaluate the impact of the newly generated datasets on the performance of a binary classifier. Despite the instability of our classifier, we were able to conclude that GAN augmented data can improve the classifier behavior.

Keywords — Classifier, GAN, data augmentation, performance

I. Introduction

Data scarcity is a common problem in machine learning applications as they often require very large datasets in order to achieve good results. Data incompatibility is another problem that arises when the application data is somewhat different than the training images. In both cases, the model is prone to overfit the training data and perform poorly in the application. As a tentative to solve this issue, we test the impact of data augmentation using Generative Adversarial Networks (GANs) on a classification problem: cows vs. horses. GANs study the distribution of an image and use it to generate new images from noise. They can therefore complexify the training data by generating new images that resemble the original ones and fill in the data distribution gaps. Data augmentation using GANs has proved itself to be efficient in literature [1],[2]. In this paper, we test the cow vs. horse classification problem, first, using the small original dataset, and second, using the GAN generated data. We compare the results using classification metrics such as loss, accuracy, precision and recall. We then demonstrate how GAN augmentation can improve the classification results.

II. Related Works and Applications

Recent literature has been focusing on improving data augmentation techniques for computer vision applications. Although classical image processing techniques involving cropping, rotating, changing the brightness, etc. work well for toy datasets and problem statements, there can be sometimes huge gaps in the dataset distribution that these classical techniques cannot fill. More advanced techniques are therefore developed to overcome this problem. State of the art methods include Auto Augment which involves searching for sub-policies to find the best combination of classical data augmentation operations to apply to a dataset. Feature augmentation is another state of the art method that consists of augmenting the distribution of the feature space rather than focusing only on the input space [4]. However, despite the efforts, these techniques have only been able to improve the accuracy of the model by 2-5% in literature. A more promising data augmentation technique involves using Generative Adversarial Networks allowing an improvement of accuracy of around 10%.

Generative Adversarial Networks (GANs) have been extensively used for data augmentation over the past few years. GAN-based data augmentation can increase the diversity of the training dataset, which in turn can improve the generalization ability of the model. By generating new synthetic samples, the GAN can fill the gaps in the training data and capture a wider range of variations in the data distribution [2]. GAN-based data augmentation can therefore fix the imbalance problem of the dataset classes such as in Andresini et al.'s classification of network traffic model [1]. It also addresses the data scarcity problem and helps in minimizing the use of sensitive original data by generating synthetic ones such as in medical applications [3].

While machine learning problems such as classification usually require a large dataset in order to achieve good results, successful GAN models such as Röglin et al.'s have validated the use of GANs with datasets consisting of less than 50 images [3]. We therefore employ GAN to augment our cows and horses dataset and improve our classification results.

III. Methods

In this section, we discuss our GAN model, our classification model, and their respective architectures:

A. Deep Convolutional Generative Adversarial Networks (GANs):

1) Overview:

The GAN is made up of two distinct models, the generator and the discriminator that work together in order to generate new images. The generator takes a noise vector *z* and spawns 'fake' images that look like training images. The discriminator then takes the newly generated image and classifies it as real or fake.

During the training, the goal of the discriminator is to maximize the probability that it correctly classifies the real and fake images by maximizing the function:

$$L_D = \log\big(\mathrm{D(x)}\big) + log(1 - D(G(z)) \quad (1)$$

Where x designates the data of an image, D(x) is the probability that x came from the training data rather than from the generator, and (1 – D(G(z)) is the probability that G(z) is a generated 'fake' image.

The goal of generator *G* is to estimate the distribution p_{data} from which the trainset comes from and generate new images with respect to this distribution. It consistently tries to outsmart the discriminator and, maximize the function:

$$L_G = log(D(G(z)) \quad (2)$$

Where D(G(z)) is the probability that the discriminator would classify the output of *G* as a real image.

The loss of the GAN model is therefore:

$$GminDmax\, V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1 - D(G(z)))] \quad (3)$$

The generator and discriminator are trained stochastically until their losses stop changing, the discriminator stagnating at an accuracy of 50% and the generator creating good images.

2) Model Architecture:

We implement a Deep Convolutional GAN which is an extension of the traditional GAN; however, it strictly uses convolutional layers. We discuss below the generator and discriminator architectures:

a) Generator:

We feed the generator a noise vector z of length *nz* = 10 using the normal distribution. We use Transpose Convolutional layers to upsample the input noise and generate 3 x 64 x 64 images. The size of the input and hidden layers decreases exponentially where *ngf*, the size of the feature map (in our case 64), is multiplied by exponentially decreasing numbers. After each one of these first four layers, we use batch normalization and *ReLU* activation. Batch normalization improves the convergence and stability of the learning process by computing the mean and variance of each channel over the batch, and then subtracting the mean and dividing by the standard deviation for each element in the channel. The three hidden layers can be then represented by:

$$h^{[i]} = ReLU(batch\_norm2d(W^{[i-1]}h^{[i-1]})) \quad (4)$$

Where h denotes the noise and W denotes the weights. The output is finally generated through a transpose convolutional layer with 3 feature maps of size 64 x 64 representing the 3 channels of the RGB image and a *tanh* activation function. It can be represented by:

$$output = tanh(W^{[o]}h^{[o]}) \quad (5)$$

The kernel size is set of 4. The stride is set to 1 in the input layer and 2 afterwards. The padding is set to 0 in the input layer and 1 afterwards. We set them this way to obtain the proper image size at the end.

b) Discriminator:

We feed the generator with 3 x 64 x 64 input images. We use 5 convolutional layers and apply *LeakyReLU* activation for the first four layers with a negative slope of 0.02. These first four layers have exponentially growing sizes, inversely to those of the generator. Their kernel size is 4, stride 2, and padding 1. We use batch normalization for the 3 hidden layers which can be summed up as:

$$h^{[i]} = LeakyReLU(batch\_norm2d(W^{[i-1]}h^{[i-1]})) \quad (6)$$

The output layer generates one value which is the probability of an image belonging to the training set and is activated by a sigmoid function. Its kernel size is 4, stride 1 and padding 0.

$$output = sigmoid\big(W^{[o]}h^{[o]}\big) \quad (7)$$

B. Classifier

Our classifier is meant to classify between two classes, in our case: cows and horses. Because of the small size of our dataset, we simplify its architecture to prevent it from overfitting. It takes a 3 x 64 x 64 image and inputs it into 2 convolutional layers (of size 16 and 8 and kernel size 5 and 3 respectively) activated by *ReLU* and followed with max pooling, and 2 dense fully connected layers. The output layer is a dense layer, outputs 2 values and is activated by the *softmax* function.

IV. Experiments and Results

A. Dataset:

Our dataset consists of 3 x 256 x 256 images of two classes: cows and horses. Each class has 82 images divided into train and test images whereby 41 images of a cow / horse taken from different angles constitute the train set and 41 images of another cow / horse constitute the test set. We use the dataset as is to evaluate the classification model before data augmentation. For both the classification and GAN models, we resize the images to 3 x 64 x 64 to reduce the computational complexity and be able to run our programs on the GPU provided by Google Colab. Small batches taken from the horses and cows train and test folders are shown in Fig. 1 below.

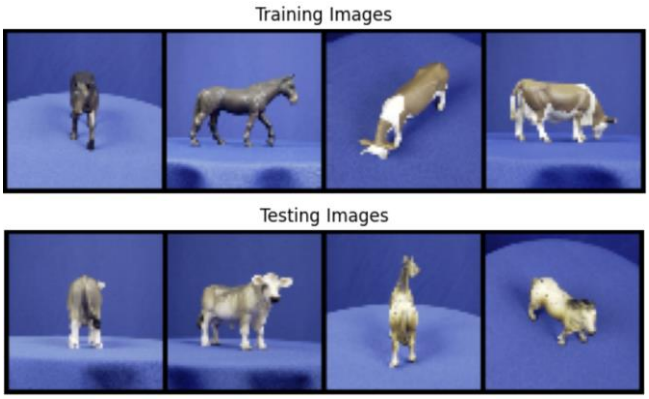

Fig. 1. Batches of size 4 taken from the test and train datasets.

B. *DCGAN Experiments and Results:*

We trained our GAN model using Adam optimizers with learning rate 0.0002 for both the generator and discriminator and Binary Cross Entropy as loss criterion. We used the train images to generate the augmented data for the cows and horses and trained the model separately for each class. The number of epochs and batch size were determined upon experimentation. Despite building our model to perform batch stochastic learning, we discovered that setting a batch size of 1 and hence performing a fully stochastic learning allows the GAN to generate images with better quality. We observe in Fig. 2 below the quality difference of generated images after 200 epochs for batch sizes of 1, 4 and 16. For a fully stochastic learning, the images are ready to use after 200 epochs. However, we discovered that increasing the number of epochs would help the generator create more versified orientations of the animal. We therefore set the number of epochs to 1000.

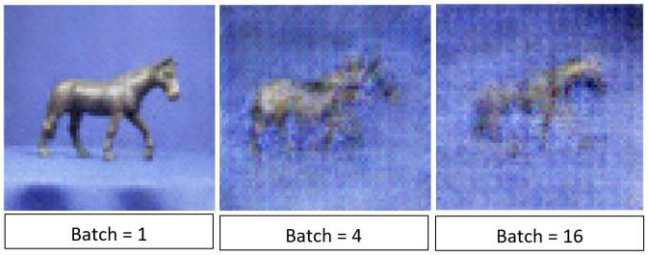

Fig. 2. Generated horse image after 200 epochs for different training batch sizes.

We plot the generator and discriminator losses vs the number of iterations and notice that after 20000 iterations the loss stagnates around 5 for both, suggesting that the generator is being able to fool the discriminator and that the discriminator is no longer able to differentiate between real and generated images (Fig. 3). Noticing that each training generates a limited number of new orientations, we augmented our data by batches generating new orientations at each run. Some of the generated pictures are shown in Fig. 4.

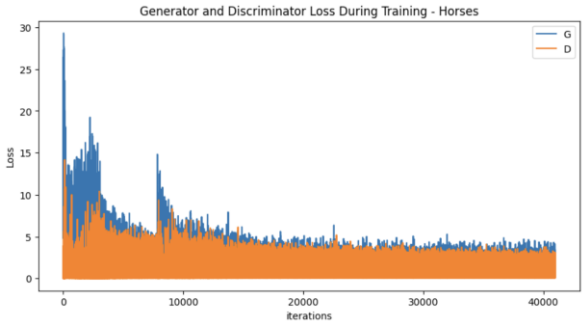

Fig. 3. Generator and Discriminator Losses (1000 epochs)

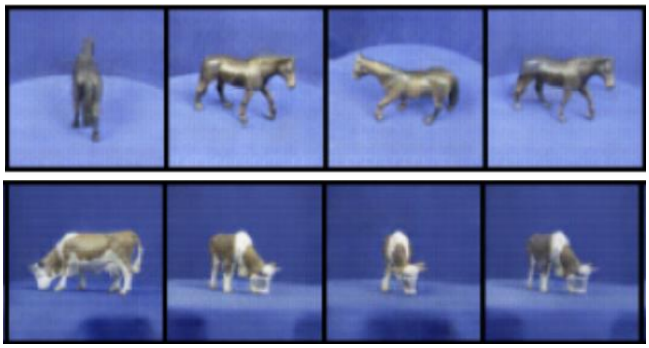

Fig. 4. Small batch of generated cows and horses

C. *Classification Experiments and Results:*

The classification task of cows and horses was carried out in 5 situations: 1) using the original dataset, 2) using the *torchvision* transforms to augment the data, 3) using the GAN augmented data + 1000 images, 4) using GAN augmented data + 2000 images, 5) using GAN augmented data with transforms. We used the following metrics to evaluate the performance of our classifier: loss, accuracy, precision, recall and f1 score. Because of the size of our dataset and the absence of a validation set, we evaluated these metrics on our test set to judge the performance of our model. While computing the metrics for our test set, we froze the model weights. For our experiments, we used a batch size or 4 and varied the epochs with a maximum number of 80. We used cross entropy as our loss criterion.

1) *Training using the original dataset:*

After training our model for 80 epochs, we got the following results (Fig. 5). We notice that our model was overfitting the train data, minimizing the loss for the train set and increasing it for the test set while reaching an accuracy of 100 % for the train set and reducing its accuracy when it comes to test data (50%). This result was expected as the used dataset was very small compared to typical datasets generally used in classification. Taking a look at the graphs, we notice that we have a higher chance of achieving higher accuracy if we train our model with a lower number of epochs. After retraining our model, we were able to achieve test accuracy of 59% after 40 epochs. Despite its simple architecture, our model still overfits the data when it comes to such a small dataset. We therefore try to augment data and test its performance.

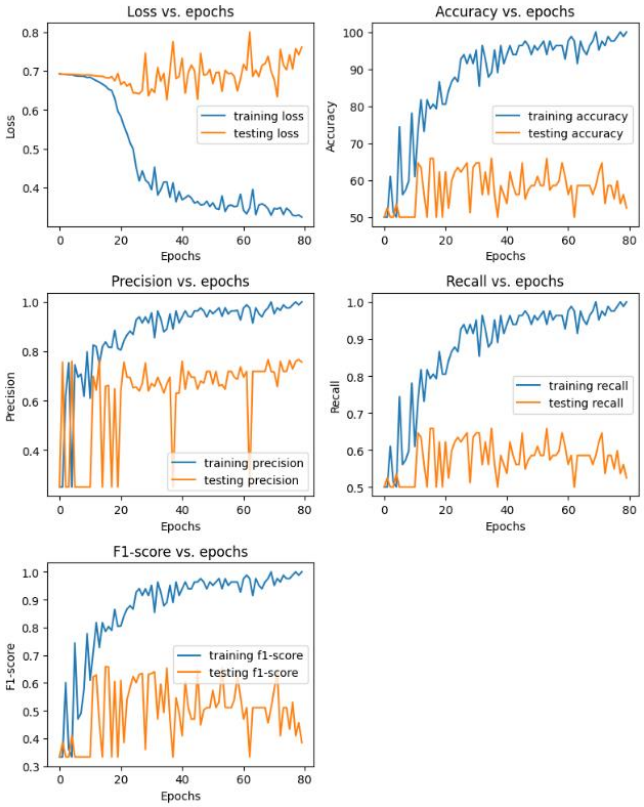

Fig. 5. Classification metrics: Training on the original dataset

2) Training using the torchvision transforms to augment the dataset:

Taking a look at the train and test data, we noticed that in both cases, for the cow and for the horse, the animal used for the test is different than the one we use to train. They have different colors and different patterns but similar shapes. An intuitive approach was to try to make the animals in the train set look more like the animals in the test set by applying transforms such as changing the color and brightness of the picture, and applying Gaussian blur to remove the pattern details. We also applied rotation in order to help the model learn the class no matter the orientation of the picture and normalize the data. As the transforms were applied randomly on the random batches in each epoch, the model saw the data as augmented despite using the same 82 pictures for the training process. A batch of our newly generated data using transforms is shown in Fig. 6.

Fig. 6. Batch of training data with transform

We noticed however that this method only added complexity to the model as both its train and test accuracy dropped and stagnated. It did not help in recognizing the features that would distinguish cows from horses. With the small number of images we had, the data became too complex for our simple classifier. We probably require a bigger network for better results.

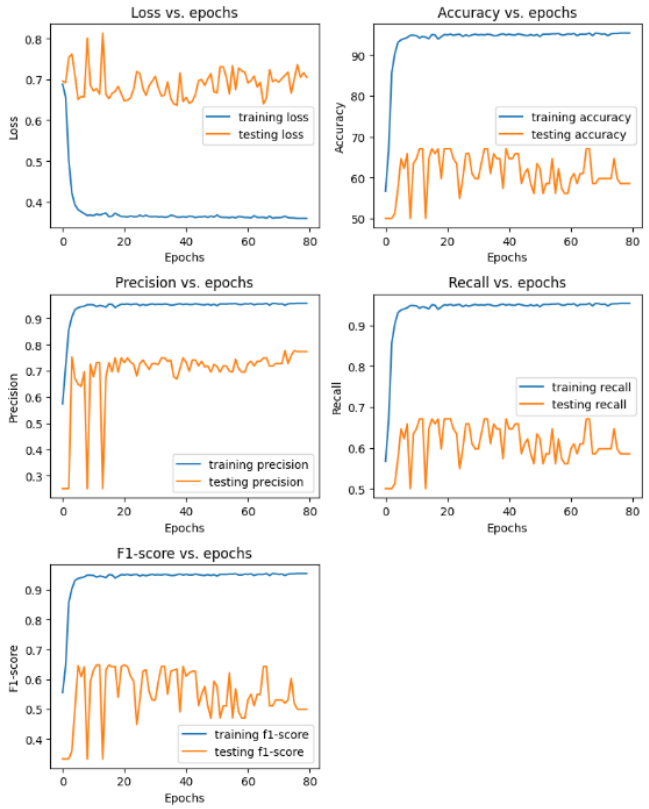

Fig. 7. Classification metrics: Training on the GAN augmented data (+ 1000 images)

3) Training with GAN augmented data + 1000 images:

Using GAN, we generated 1000 new images (500 cows, 500 horses) and used them to train our classification model. Despite noticing that a smaller number of epochs would be sufficient to reach the same results, we chose a large number of epochs to better visualize the behavior of the model during the training process. Looking at the loss curves, we can still see the model overfitting the data but to a lesser extent the original set. We also observe an upward shift of the accuracy curve compared to that of the training using the original dataset. In the first training run, we reached an accuracy of 59 %. We notice from the accuracy curve that we have higher chances of reaching improved results by varying the number of epochs than with the original dataset (Fig. 7).

4) Training with GAN augmented data + 2000 images

Using GAN, we generated 2000 new images (1000 cows, 1000 horses) and used them to train our model. With the 1000 new images, we notice that our model was still overfitting the data (Fig. 8).

5) Training with GAN augmented data + 2000 images and applying transforms:

Applying the transforms to the 2000 generated images did not improve the model, but rather only caused more fluctuations in the loss function.

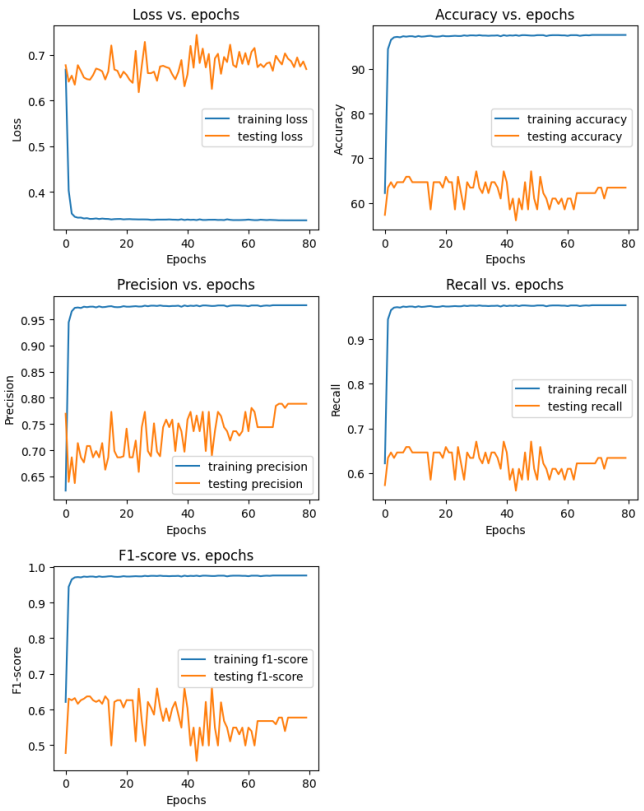

Fig. 8. Classification metrics: Training on the GAN augmented data (+ 2000 images)

V. Discussion:

The summary of our results can be shown in Table 1. In all the cases, our model was overfitting the data. We noticed a slight improvement with GAN data augmentation with an increase of test accuracy ranging between 4 and 13%, which agrees with literature results. However, we cannot get precise evaluations of our metrics as our training process lacks stability. This problem can be related to many factors in our training process:

- The images were resized to 3 x 64 x 64 which made us lose information about the images and caused the model to overfit.
- The train set is different than the test set. Augmenting the train dataset did not significantly improve our model.

TABLE I. Summary of Classification Metrics

Metric	Original		Transform		GAN 1000		GAN 2000		GAN 2000 + transform	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Loss	0.34	0.76	0.39	0.69	0.36	0.71	0.34	0.67	0.34	0.74
Accuracy	1.0	0.5	0.94	0.58	0.95	0.59	0.97	0.63	0.97	0.56
Precision	1.0	0.76	0.94	0.64	0.96	0.78	0.97	0.78	0.97	0.58
Recall	1.0	0.52	0.94	0.58	0.95	0.59	0.97	0.63	0.97	0.56
F1	1.0	0.38	0.94	0.63	0.95	0.50	0.97	0.58	0.97	0.53

Some ways to improve this model would be to increase the complexity of the dataset using GAN augmentation while increasing the size of the model to handle such a complex dataset. A way to stabilize the training process would be to use regularization techniques. Alternative GAN architectures can also be used for a better complexification of the data.

VI. Conclusion

After optimizing our GAN and classification models, we were able to study the impact of GAN data augmentation on the performance of the classifier. We discovered that the GAN is sensitive to the batch size and the number of epochs while the classifier is more sensitive to the nature of the dataset. Despite the instability of our classification model, we were able to deduce that GAN data augmentation is in general helpful in improving the classifier's performance.

Acknowledgment

We acknowledge the guidance of Prof. Hedi Tabia and the advice of our classmates in completing this project.

References

[1] Andresini, Giuseppina, et al. "Gan Augmentation to Deal with Imbalance in Imaging-Based Intrusion Detection." *Future Generation Computer Systems*, vol. 123, 2021, pp. 108–127., https://doi.org/10.1016/j.future.2021.04.017.

[2] Goodfellow, I., et al. "Generative adversarial nets." Advances in neural information processing systems (2014).

[3] Röglin, Julia, et al. "Improving Classification Results on a Small Medical Dataset Using a Gan; an Outlook for Dealing with Rare Disease Datasets." *Frontiers in Computer Science*, vol. 4, 2022, https://doi.org/10.3389/fcomp.2022.858874.

[4] Yang, Suorong et al. "Image Data Augmentation for Deep Learning: A Survey." Nanjing University, 2022. https://doi.org/10.48550/arXiv.2204.08610