Synthetic Data Augmentation to Aid Small Training Datasets

Final Report 11/17/2020

Mehrad Khalesi
mkhalesi@stanford.edu

Shobit Kishore
shobit@stanford.edu

Stephen MacCabe
smaccabe@stanford.edu

1 Introduction

Our project goal is this: To explore generative data synthesis as a practical technique for aiding small training dataset situations. If this project succeeds, we will have proven a way to allow some deep neural network classifiers to perform at a state-of-the-art level, while consuming a fraction of the positively labeled data required by a standard approach.

This is an experimental investigation into using Generative Adversarial Networks (GANs) to enhance the effectiveness of training deep neural network classifiers, thus extending the reach of these classifiers in situations where labeled data is scarce or expensive. We are employing a GAN to create artificial images with a known label. We then combine these images with a small number of actual labeled images to satisfy a classifier network's training needs, while using only the smaller set of the actual labeled data. Thus, the classifier, ingesting the enhanced input data (actual and generated images) will be able to classify unseen data effectively.

The expense and limited availability of labeled data comprises a severe constraint on supervised learning applications, especially deep neural networks. These systems' appetite for labeled data is the choke point in a number of cases. There has been work to solve this problem such as adaptive discriminator augmentation [1], and our work is a completely separate approach - supplying more examples at very low 'cost'.

2 Code Repository

We maintain our code on github, in our shared repository, with an easily-identifiable directory structure including data, Jupyter top-level models with notebooks, and Python utilities. The repository is located here: `https://github.com/mehrad-kh/cs230-project`

3 Data

We are using the 'Planes in Satellite Imagery' dataset from Kaggle as our vehicle. Each image is a 20 X 20 pixel, 3-color picture from above, containing either a plane or just empty earth. The data contains 8,000 images (20 x 20 x 3) labeled as 'plane' and 24,000 images tiles from the 'no-plane' class.

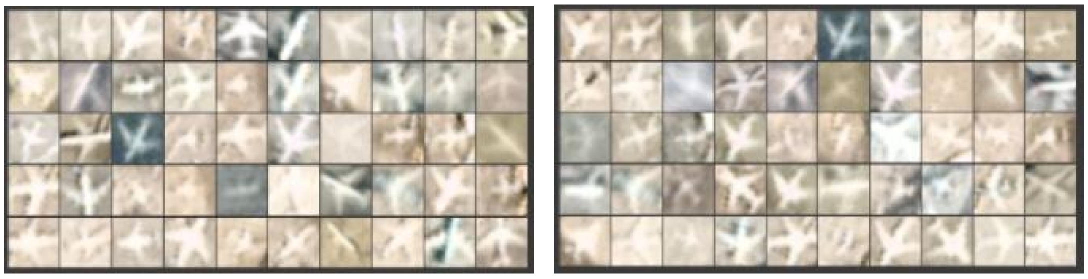


Figure 1: Two grids containing multiple sample positive images from 'Planes'

We chose this dataset for several reasons. First, it is freely available from Kaggle without privacy hassles. Second, the images are small images, and at inception of our effort it was reasonable for us to expect our GAN to perform well on these low-dimensionality inputs.Third, a large number of labeled images are available–one part of the investigation is to determine how many labeled images we need in order to perform effectively when they are combined with our

GAN-generated images. As it turned out, we are able to demonstrate the hoped-for effect with our initial choice of 800 positive examples.

4 Our Baseline Data Usage

Since our goal is to improve performance of classification with a small number of labeled examples, we chose a small fraction (800 of the original 8,000) of positive images, and a large fraction (16,000 of the original 24,000) of negative images for training. An area we anticipate investigating is pushing down the 800 number further, to see how much leverage we can get. Initially, we have split 1200 positive examples and 24,000 negative examples of the Planes dataset into training, validation and test sets as shown below.

Dataset	Plane (Positive)	Non-Plane (Negative)
Trainig Set	800	16000
Validation Set	200	4000
Test Set	200	4000

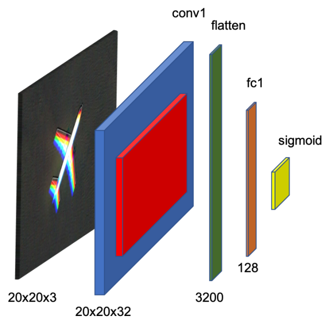
Table 1: Amount of data allocated to training, validation and test sets

5 Standalone Classification Model Architecture

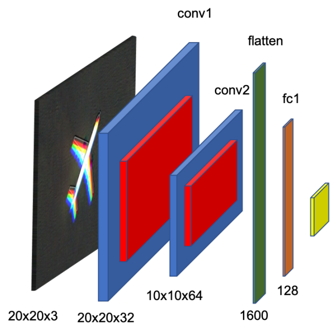
A baseline model establishes the minimum model performance with initial training dataset to which all our future experiments with synthesized data will be compared. For our model we have decided to use a convolutional neural network with general architecture principles of VGG16 model [2]. The architecture involves stacking convolutional layers with 32, 64, 128 (3 x 3) filters followed by a max pooling layer. These layers together form a block. We have implemented 3 classifiers using 1, 2 and 3 VGG blocks followed by a fully connected layer and an output layer with sigmoid activation function. The other layers use the Relu activation function and the He weight initialization. We fit the model with stochastic gradient descent; We used learning rate $\alpha = 0.001$ and momentum parameter $\beta = 0.9$. For the baseline, the model was trained on 800 plane (positive) and 16000 no-plane (negative) images. The model classification accuracy, recall and F1 score were evaluated on both training and test dataset. We decided to measure recall and F1 score given the imbalanced nature of our dataset in which negative examples outnumber positive ones.

Dataset	Training Set			Test Set		
	Accuracy	Recall	F1 Score	Accuracy	Recall	F1 Score
Model 1 Baseline	96.56	35.89	42.77	96.40	31.21	39.57
Model 2 Baseline	96.76	36.75	44.10	96.54	31.91	38.87
Model 3 Baseline	97.30	43.19	51.66	96.85	34.74	43.68

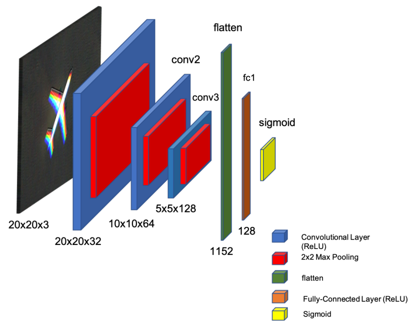
Table 2: Baseline Performance



(a) Model 1



(b) Model 2



(c) Model 3

Figure 2: VGG16 based classifier model architectures

6 Generating Synthetic Training Images

6.1 Vanilla GAN

In order to enhance our baseline training data, the first GAN architecture we experimented with is the original Vanilla GAN implementation by Ian Goodfellow et al. [3] We implemented our own Vanilla GAN from scratch to transform a noise vector of dimension (100,1) to an image of dimension (20,20,3). Below are some generated samples from the Vanilla GAN.

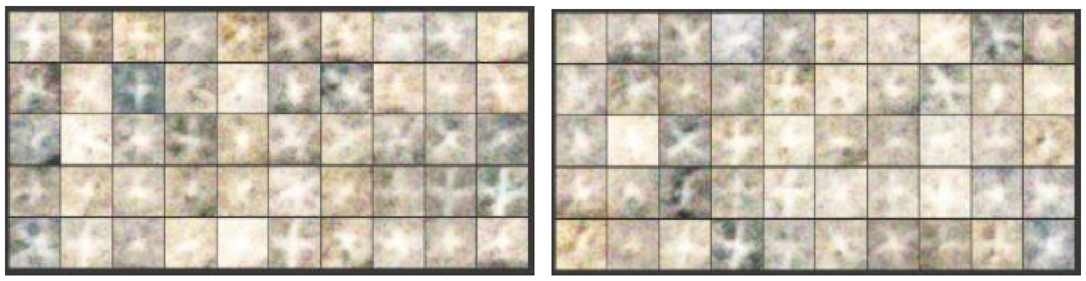


Figure 3: Sample positive images generated by the Vanilla GAN

The Vanilla GAN learns the underlying distribution of the pixels well enough to be considered somewhat plausible to the human eye. There is however presence of some pixelated noise and a general loss of sharpness in the outputs. The FID score [4] for this GAN variant was the highest among all variants. We noted a drastic improvement in the FID score with convolution based GAN architectures noted below and decided to spend more time experimenting with and improving them.

6.2 DCGAN

The second implementation we are experimenting with is the DCGAN [5].This GAN variant generates data outputs that look significantly better than the Vanilla GAN outputs. The noise artifacts that the Vanilla GAN produces are absent here. The synthesised images look more real, with some synthesised images looking just as real as the original data. The FID score for this GAN is the lowest of all variants.

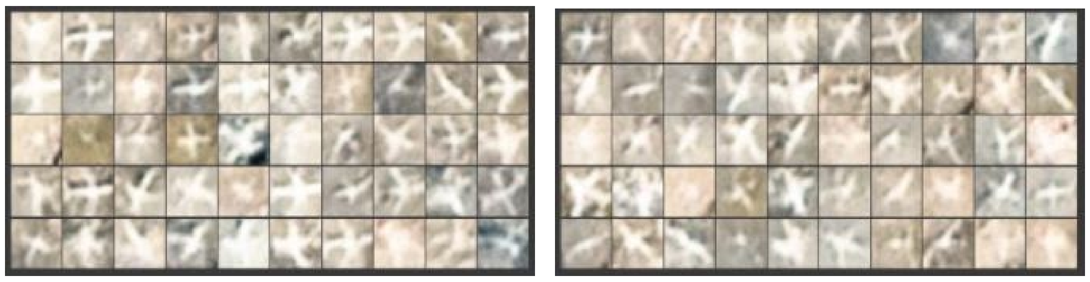


Figure 4: Sample positive images generated by the DCGAN

6.3 WPGAN

The third implementation we are experimenting with is the WPGAN [6]. With this variant, we also see an obvious improvement over the Vanilla GAN. The model does quite well to learn the distribution of the pixels that make a plane image and all of the variations in angles, sizes and sharpness of the planes. Again, some of the generated data looks very real and pleasing. The one thing that may adversely affect the quality of the generated images is the GAN's ability to learn the background colors. As we can see from the sample generated data, the colors depict a higher level of color saturation and contrast. This could be one reason for the WPGAN registering a higher FID score.

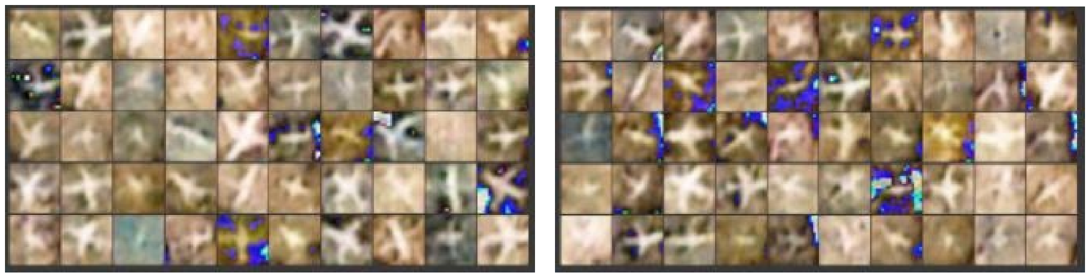


Figure 5: Sample positive images generated by the WPGAN

6.4 LSGAN

The last implementation we are experimenting with is the LSGAN [7]. The LSGAN also produces visually realistic images that rival the DCGAN in quality. These synthesised images seem to learn the distribution of the original data quite well. Along with extremely real-looking examples, we notice some that have noisy artifacts. The FID score for LSGAN outputs was slightly higher than the DCGAN outputs.

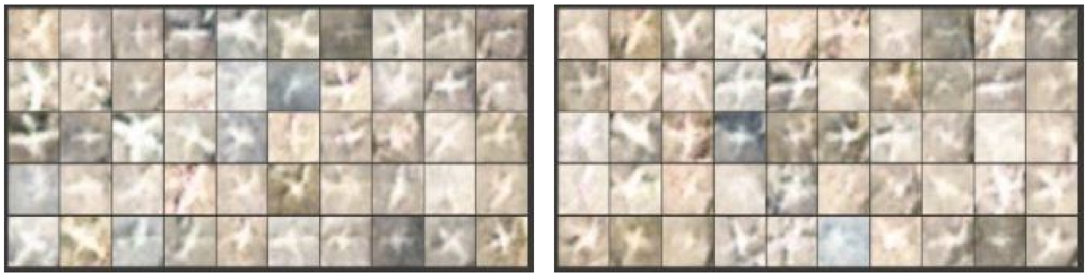
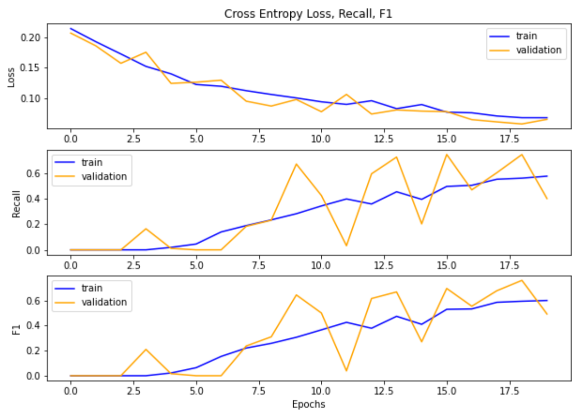


Figure 6: Sample positive images generated by the LSGAN

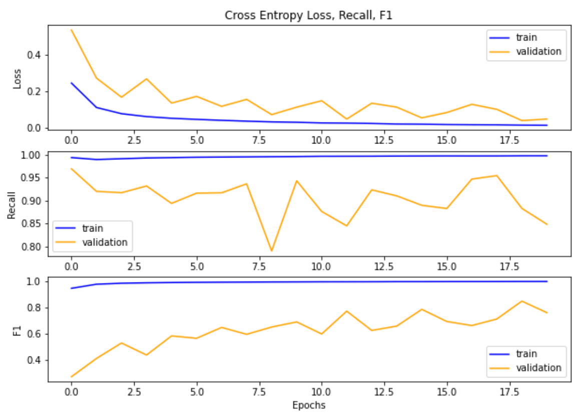
We experienced that GAN performance is very sensitive to hyperparameter changes, much more so than traditional deep learning classifiers. The training process is very volatile to slight changes in hyperparameters such as batch-size and learning rate. Additionally, we found that tracking the discriminator and the generator loss is not very intuitive and informative. In order to get the outputs we were able to get, we experimented with a different combination of batch sizes and learning rates. The batch sizes were usually a multiple of 16 ranging from 16 to 64 and learning rates a multiple of 10 ranging from 1e-5 to 1e-2 for different models.

7 Results & Discussion

We have implemented three classifier models and trained them with sampled synthetic data generated by Vanilla GAN, DCGAN, WPGAN and LSGAN. Figure 7 shows Model 3 (3 VGG block) Cross Entropy Loss, Recall and F1 score measured on training and validation set during the training of our model with baseline data with no additional sysnthetic data on the left and training with baseline data plus additional 80000 DCGAN synthetic data on the right. After training our model with the baseline data plus 80000 synthetic data generated by DCGAN, the recall and F1 score increased from 64.02% to 88.42% and 64.46% to 78.84 % respectively on the validation set on average across all seeds.



(a) Baseline data



(b) Baseline data plus 80000 DCGAN synthetic data

Figure 7: Model 3 Cross Entropy Loss, Recall and F1 score trained with baseline data and baseline plus 80000 DCGAN synthetic data for a single random seed

The curves in Figure 8 show the answer to our project's question: What is the effect of adding our GAN-created images to the training data? The plots reflect the recall rates and F1 score from our 3 block VGG-16 classifier. The x-axis shows the number of GAN-generated images added to our baseline classifier, ranging from 0 to 80,000. The y-axis shows the corresponding recall rates. To deal with variation in the baseline classifier, we repeated the work with multiple random seeds, seen here in different colors. We see that for the DCGAN and LSGAN, which have low FID values, the recall rates improve and converge to a high value. We can conclude that: Adding synthetic images to the training set improves the recall rates for the DCGAN and LSGAN. The term 'unqualified' in the title means we did not filter GAN images through the baseline discriminator before adding them.

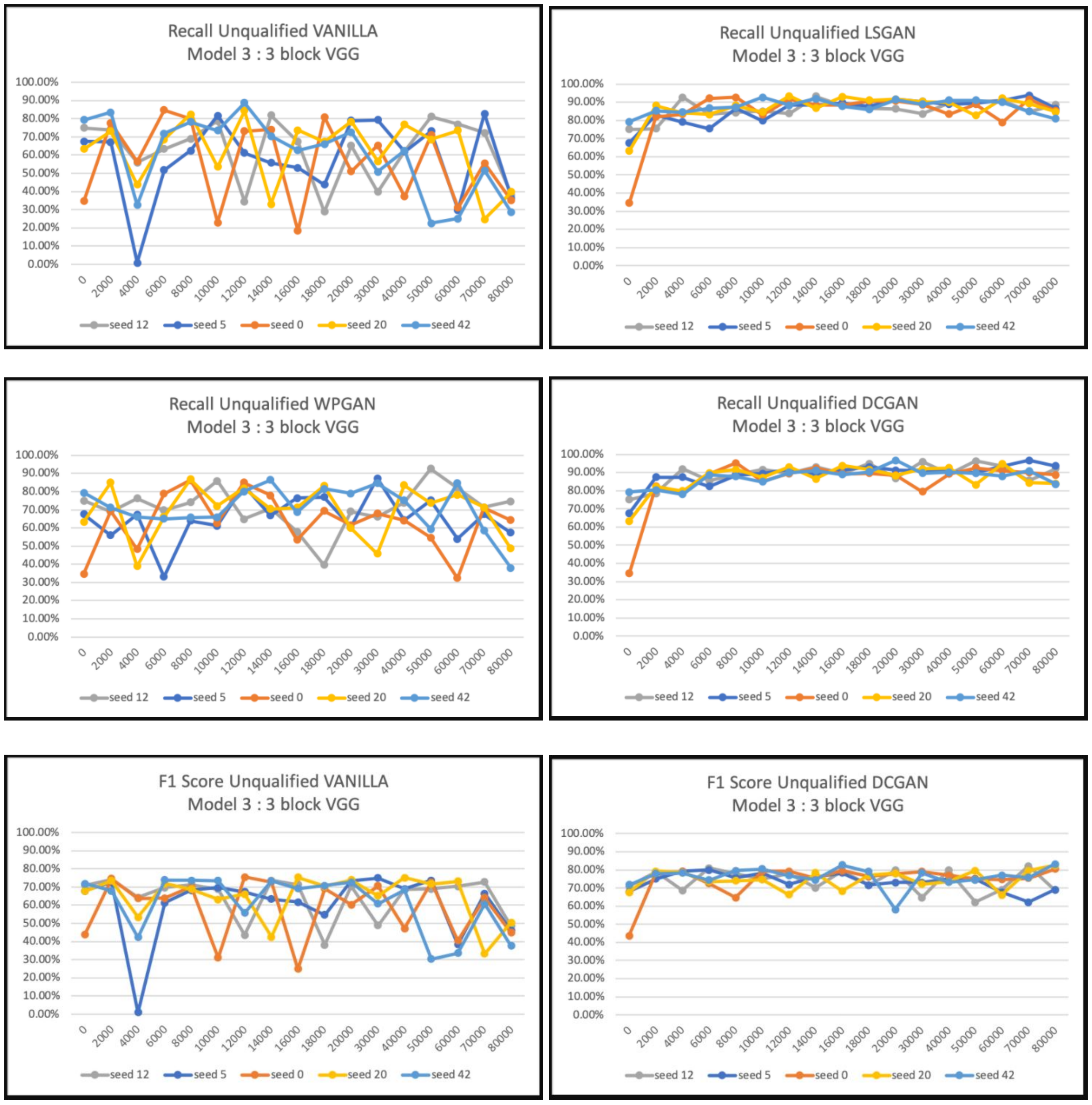


Figure 8: Test set Recall and F1 score measured on Model 3 (3 Block VGG) trained with sampled synthetic data generated by Vanilla GAN, LSGAN, WPGAN and DCGAN

8 Conclusion

We have demonstrated that there exist situations where GANs can be used to extend the training data and improve performance of a binary classifier in cases where that data is limited.

FID scores generally predict the degree of improvement to the classification metrics.

We gained no additional benefit by pre-qualifying the synthetic data.

9 Contributions

- Shobit Kishore: Concept for the project, Research into current GAN versions, Vanilla GAN authoring and exploitation,DCGAN, LSGAN optimization and explotiation, Reporting
- Mehrad Khalesi: GitHub setup and administration, Discriminator R&D - choice and implementation, Top level GAN + Discriminator investigations, Data generation, collation and presentation, Reporting
- Steve MacCabe: FID computations, Image Tiling code and utilities (unused), WPGAN exploitation, Administrative Logistics, Investigation of Conditional GAN and Discriminator Augmentation, Reporting

References

[1] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, Timo Aila, "Training Generative Adversarial Networks with Limited Data"
`https://arxiv.org/abs/2006.06676`

[2] Karen Simonyan and Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition"
`https://arxiv.org/abs/1409.1556`

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, "Generative Adversarial Nets"
`https://arxiv.org/pdf/1406.2661.pdf`

[4] Huesel, Ramsauer, Unterthiner and Nessler, "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium"
`https://arxiv.org/abs/1706.08500`

[5] Alec Radford, Luke Metz, Soumith Chintala, "Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks"
`https://arxiv.org/pdf/1511.06434v2.pdf`

[6] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville, "Improved Training of Wasserstein GANs"
`https://arxiv.org/abs/1704.00028`

[7] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, Stephen Paul Smolley, "Least Squares Generative Adversarial Networks"
`https://arxiv.org/abs/1611.04076`