Methods for Generative Adversarial Output Enhancement

Michael B. Brodie
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Physical Sciences and Mathematics Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/8763

This Dissertation is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact ellen_amatangelo@byu.edu.
Methods for Generative Adversarial Output Enhancement

Michael B. Brodie

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Tony Martinez, Chair
  David Wingate
  Bryan Morse
  Jacob Crandall

Department of Computer Science
Brigham Young University

Copyright © 2020 Michael B. Brodie
All Rights Reserved
ABSTRACT

Methods for Generative Adversarial Output Enhancement

Michael B. Brodie
Department of Computer Science, BYU
Doctor of Philosophy

Generative Adversarial Networks (GAN) learn to synthesize novel samples for a given data distribution. While GANs can train on diverse data of various modalities, the most successful use cases to date apply GANs to computer vision tasks. Despite significant advances in training algorithms and network architectures, GANs still struggle to consistently generate high-quality outputs after training. We present a series of papers that improve GAN output inference qualitatively and quantitatively. The first chapter, Alpha Model Domination, addresses a related subfield of Multiple Choice Learning, which – like GANs – aims to generate diverse sets of outputs. The next chapter, CoachGAN, introduces a real-time refinement method for the latent input space that improves inference quality for pretrained GANs. The following two chapters introduce finetuning methods for arbitrary, end-to-end differentiable GANs. The first, PuzzleGAN, proposes a self-supervised puzzle-solving task to improve global coherence in generated images. The latter, Trained Truncation Trick, improves upon a common inference heuristic by better maintaining output diversity while increasing image realism. Our final work, Two Second StyleGAN Projection, reduces the time for high-quality, image-to-latent GAN projections by two orders of magnitude. We present a wide array of results and applications of our method. We conclude with implications and directions for future work.

Keywords: Generative Adversarial Networks, image generation, multiple choice learning, deep learning, generative modeling
ACKNOWLEDGMENTS

Thank you to all those who helped and encouraged me throughout the PhD process. I want to thank especially my advisor, Tony Martinez, for his support and guidance along the way; for connecting me with opportunities in industry; and for helping me hone my research abilities through clear and prompt feedback. I also thank my committee members, Bryan Morse, Christophe Giraud-Carrier, and David Wingate, for providing feedback on paper submissions and research proposals along the way.

I thank Chris Tensmeyer, Logan Mitchell, Wes Ackerman, Taylor Archibald, and Alex Lin for help with reviewing and refining model architectures and algorithms. Many thanks to my research assistants, Steven Barnett and Brian Rasmussen, for their help in conducting and analyzing results of many, many sets of experiments.

Thank you to my parents for their wonderful examples and advice along the way. Lastly, I thank my wonderful wife, Amanda, and son, Jacob, for the joy and fun they add to my life.
# Table of Contents

List of Figures ix  
List of Tables xii  

1 Introduction 1  
1.1 Summary of Contributions 3  

References 4  

2 Alpha Model Domination 11  
2.1 Introduction 12  
2.2 Related Work 14  
2.2.1 Multiple Choice Learning 16  
2.3 Alpha Model Domination 18  
2.3.1 AMD Frequency 20  
2.3.2 AMD Prevention 21  
2.3.3 Random Selection 22  
2.3.4 Random Annealing 22  
2.3.5 Stochastic Softmax 23  
2.4 Experiments 24  
2.4.1 CIFAR-10 24  
2.4.2 Image Segmentation 25  
2.5 Conclusion 26
# PuzzleGAN

4.1 Introduction .................................................. 65

4.2 Related Work ................................................. 66
  4.2.1 Self-supervised learning ............................... 67
  4.2.2 Puzzle-based losses ................................... 68

4.3 Method ......................................................... 69
  4.3.1 Puzzler architectures and training ................... 70
  4.3.2 Self-supervised GANs .................................. 72

4.4 Experiments .................................................. 74
  4.4.1 Finetuning setup ....................................... 74

4.5 Results ....................................................... 75
  4.5.1 PGGAN ................................................... 75
  4.5.2 Amazon Mechanical Turk .............................. 76
  4.5.3 BigGAN .................................................. 77
  4.5.4 Qualitative results .................................... 80

4.6 Network visualization ....................................... 82

4.7 Conclusion .................................................. 85

References ....................................................... 87

# Trained Truncation Trick

5.1 Introduction .................................................. 97

5.2 Related ........................................................ 98
  5.2.1 Truncation trick precursors ........................... 98
  5.2.2 Truncation trick ........................................ 98

5.3 Method ........................................................ 100
  5.3.1 Trained Truncation Trick (TTT) ....................... 100

5.4 Experiments .................................................. 103
5.5 Results ................................................................. 104
  5.5.1 StyleGAN2 Quantitative Results ............................. 104
  5.5.2 StyleGAN2 Qualitative results ............................... 105
  5.5.3 PGGAN .......................................................... 108
  5.5.4 BigGAN ........................................................ 108
  5.5.5 User study ..................................................... 110
  5.6 Conclusion ....................................................... 112

References 113

6 Two Second StyleGAN Transfer 119
  6.1 Introduction ..................................................... 120
  6.2 Related .......................................................... 121
    6.2.1 Generative Adversarial Networks .......................... 121
    6.2.2 StyleGAN2 .................................................. 122
    6.2.3 Image Projection .......................................... 122
  6.3 Method .......................................................... 124
    6.3.1 Fast StyleGAN2 Projection ................................. 124
    6.3.2 Properties of FSP latent space ........................... 126
    6.3.3 Inverted Layer Importance ............................... 127
    6.3.4 Image Normalization ...................................... 129
    6.3.5 Training ................................................... 130
  6.4 Experiments ..................................................... 131
    6.4.1 Image Interpolation ...................................... 131
    6.4.2 PCA Interpolation ........................................ 134
    6.4.3 Style Transfer ............................................ 135
    6.4.4 Quantitative Reconstruction Quality ..................... 136
  6.5 Conclusion ..................................................... 137
References 140

7 Conclusion 148
# List of Figures

2.1.1 Example of Alpha Model Domination in the CamVid semantic segmentation task. .................................................. 13

3.1.1 CoachGAN applied to GAN trained on CelebA. ......................... 38

3.3.1 Visualization of CoachGAN optimization path. ....................... 43

3.3.2 Histograms of pixel and latent distances. .......................... 45

3.3.3 Comparisons of the original and coached samples. ............... 46

3.3.4 Gradual transition of outputs via CoachGAN. ..................... 46

3.3.5 CoachGAN applied directly to pixel output. ....................... 47

3.4.1 Effects of different learning rates. ................................ 49

3.4.2 Transition from original image to final CoachGAN output. ......... 50

3.4.3 5-Nearest Neighbors in the training set. .......................... 50

3.4.4 CoachGAN output using LSUN bedroom dataset. ............... 51

3.4.5 Original and CoachGAN samples using PGGAN. .................. 52

4.1.1 Comparison of generated outputs from pretrained (left) and finetuned with PuzzleGAN (right) BigGAN-deep models. The PuzzleGAN output exhibits a more natural focus on the image center, while reducing background clutter and increasing continuity of table edges. .................................. 65

4.3.1 Overview of GAN training with PL involving a Generator (G), Discriminator (D), Gamemaker (GM), and Puzzler model (P). ............... 70

4.3.2 Comparison of DCGAN-based 16-piece Puzzlers trained for 72 hours on ImageNet data with input sizes 64x64, 128x128, and 256x256. ................. 71
4.5.1 Comparison of DCGAN (top) and ResNet (bottom) BigGAN finetuning with 
\[ P_4 + P_{16} + D. \] ................................................................. 79
4.5.2 Finetuning PGGAN with PuzzleGAN. .................................................. 80
4.5.3 Finetuning BigGAN with PuzzleGAN. .................................................... 81
4.5.4 PuzzleGAN samples using pretrained BigGAN-deep model. .................. 83
4.5.5 Additional PuzzleGAN samples using pretrained BigGAN-deep model. .... 84
4.6.1 PuzzleGAN saliency visualizations. ...................................................... 85

5.3.1 TTT overview. ..................................................................................... 101
5.3.2 Architecture variations of TTT explored in this work. ......................... 102
5.5.1 StyleGAN2 car samples using no truncation (original), TTz, TTTz, TTw, 
and TTTw. ................................................................................................. 106
5.5.2 StyleGAN2 FFHQ samples using no truncation (original), TTz, TTTz, TTw, 
and TTTw. ................................................................................................. 107
5.5.3 StyleGAN2 cat samples using no truncation (original), TTz, TTTz, TTw, 
and TTTw. ................................................................................................. 107
5.5.4 Comparison of the effects of the number of TTTz layers and the number of 
training iterations. ..................................................................................... 109
5.5.5 Illustration of slider task with 95% confidence interval results for the content 
preservation user study. ........................................................................... 111

6.1.1 Fast StyleGAN2 Projection Examples. ............................................... 121
6.2.1 Comparison of different kinds of StyleGAN2 projections. .................... 123
6.3.1 Comparison of original images and fast projections. ........................... 127
6.3.2 Importance of high-quality WN initialization. ...................................... 128
6.3.3 Edge details embedded into the 7th noise map. ................................... 128
6.3.4 Movement along the first PCA components of several noise maps. ...... 129
6.3.5 The magnified regions illustrate the noise frequently introduced by SOTA projection methods using convolution-based GANs. 131

6.4.1 Interpolations between $W^+N$ projections of LSUN bedroom (top) and CelebA (bottom) samples. 132

6.4.2 Interpolations between FSP projections of LSUN bedroom (top) and CelebA (bottom) samples. 133

6.4.3 Interpolations between PCA decompositions over FSP projections of the CIFAR-10 test set. 134

6.4.4 Interpolations between PCA decompositions over FSP projections of the Magpie subset of ImageNet. 135

6.4.5 Style transfer using PCA interpolation with $W$ held constant. I.e., we use $W$ from the right FSP projection for all image reconstructions. This transfers some of the color and texture details from the far right image to other interpolated image reconstructions. 135

6.4.6 Projection comparisons of Cityscapes, CelebA, and LSUN Bedroom. 138
List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.1</td>
<td>CIFAR-10 sMCL Accuracy and Oracle Accuracy</td>
<td>19</td>
</tr>
<tr>
<td>2.3.2</td>
<td>CIFAR-10 sMCL Number of Parameter Updates</td>
<td>19</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Results for 10 trials using the CIFAR-10 and ImageNet datasets. Average final accuracy and proportion of models experiencing AMD is shown.</td>
<td>20</td>
</tr>
<tr>
<td>2.3.4</td>
<td>CIFAR 10 Runs</td>
<td>20</td>
</tr>
<tr>
<td>2.3.5</td>
<td>ImageNet Runs</td>
<td>20</td>
</tr>
<tr>
<td>2.4.1</td>
<td>CIFAR 10 Oracle Accuracy at 5000 Epochs</td>
<td>24</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Averaged results for 5 trials of 4-member sMCL and RAR ensembles on the CamVid dataset.</td>
<td>26</td>
</tr>
<tr>
<td>2.4.3</td>
<td>sMCL</td>
<td>26</td>
</tr>
<tr>
<td>2.4.4</td>
<td>RAR</td>
<td>26</td>
</tr>
<tr>
<td>2.6.1</td>
<td>Expanded CIFAR 10 Oracle Accuracy at 5000 Epochs</td>
<td>27</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Expanded results for CIFAR-10 and ImageNet datasets. Each trial reports the final accuracy and if AMD occurred during training (+).</td>
<td>28</td>
</tr>
<tr>
<td>2.6.3</td>
<td>CIFAR 10 Runs</td>
<td>28</td>
</tr>
<tr>
<td>2.6.4</td>
<td>ImageNet Runs</td>
<td>28</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Inception Score calculated before (PRE) and after CoachGAN (POST), where higher scores are better.</td>
<td>55</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Frechet Inception Distance calculated with $z$ samples before (PRE) and after applying CoachGAN (POST). CoachGAN improves FID (lower is better) for both CelebA and CIFAR-10 pretrained DCGAN models.</td>
<td>56</td>
</tr>
</tbody>
</table>
3.4.3 Generation runtime (in seconds) with and without CoachGAN.

4.5.1 FID Results for PuzzleGAN finetunings of PGGAN (lower is better).

4.5.2 AMT Results for PGGAN. Each table entry indicates the average slider value with the row image on the left and column image on the right. Values below 50 signify that humans prefer the row method (and vice versa).

4.5.3 Left: FID results for BIGGAN (lower is better). Right: IS results for BIGGAN (higher is better). We score metrics using 50,000 samples with G models finetuned for 25k-100k iterations and report the best result across four model snapshots.

5.5.1 StyleGAN FID results

5.5.2 Progressive Growing of GAN FID results. We evaluate them against the FID statistics computed over FFHQ, since CelebA-HQ only contains 30k images.

5.5.3 BigGAN FID results

5.5.4 Mean slider values with 95% confidence intervals for (left) realism and (right) content preservation user studies on Amazon Mechanical Turk.

6.3.1 Mean ($\mu$) and standard deviation ($\sigma$) of the first four consecutive dimensions of W for fourteen scene images projected using FSP (the bottom row of Figure 6.3.1 shows example images). The means vary widely, while standard deviations show comparatively small differences.

6.4.1 Quantitative comparison of $W^+N$ projection and FSP. For LPIPS we report both AlexNet (left) and VGG (right) scores in the same column.
Chapter 1

Introduction

Generative learning comprises a vast range of approaches for synthesizing novel samples in one or more target domains. During the advent of deep learning, advances such as deep convolutional networks [26], dropout [41], improved optimization [24, 49], residual architectures [10, 18, 19, 26], gradient stabilizing activations [11, 31, 47] and regularization [13, 15] steadily transferred from discriminative approaches to enable noted but modest improvements in generative learning. The introduction of Generative Adversarial Networks [14], however, significantly enhanced the output quality of generative models.

The original GAN formulation places a generator, \( G \), and discriminator, \( D \), against one another in a two-player, zero-sum minimax game. In this setting, \( D \) aims to distinguish between real samples and generated data. The generated data comes from \( G \), which maps inputs from a latent probability space, \( p(z) \), to approximate samples in the target domain, \( p_{data}(x) \). This leads to the following loss formulation,

\[
\min_G \max_D \mathbb{E}_{x \sim p_{data}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))] ,
\]

where \( G \) and \( D \) seek respectively to minimize or maximize the overall loss. Although \( G \) and \( D \) theoretically converge to a Nash Equilibrium [14], limitations in model capacity and a non-convex gradient descent optimization can lead to mode-collapse [40] or sub-optimal solutions [32].

Impressive results in unconditional and conditional image synthesis [14, 20, 21, 29, 30, 33, 37, 45, 51, 52], image stylization [9, 35, 48] and editing [16, 36, 44], image superresolution
and video generation [1, 8, 42, 43, 46, 50] has created a vibrant GAN subfield in generative machine learning. Although computer vision applications dominate the research landscape, other domains such as music [12] and biology [34] have produced state-of-the-art results in their respective fields by adopting GAN models or adversarial training approaches.

Early improvements in GAN literature focused on heuristic training improvements [39] to avoid failed training or mode collapse, where $G$ generates a single or limited range of synthetic outputs. Parallel work introduced improved GAN losses with more stable gradient flows [2, 3, 17, 28]. Despite these improvements, successful GAN training remains a difficult task of balancing hyperparameters, loss functions, regularization schemes, and model capacities.

Recent work yields improvements in GAN output quality by introducing novel architectures. BigGAN [4] leverages vast amounts of data, compute resources, and deep layer architecture to improve conditional ImageNet [38] generation. The Progressive Growing of GANs [21] model uses a gradual training scheme to enable realistic high-resolution image synthesis. The subsequent StyleGAN family of models [22, 23] introduce a highly-engineered, style-transfer-inspired network. Their contributions include an input mapping network that transforms latent input $z$ to a more linear-like latent space, $w$. StyleGAN also modulates (i.e., ‘stylizes’) hidden output activation channels by using weights conditioned upon the original latent input, $z$. These changes, in addition to other architecture improvements, allow $G$ to distangle and interpolate the contents and styles of output images.

New discriminative learning algorithms, for example, the human pose estimation approach in [25] or the panoptic segmentation method in [27], increasingly use pretrained weights (i.e. from a deep ResNet or other successful network architectures trained on ImageNet) as a feature extraction backbone for novel network layers. As architectures grow in complexity and required training time, the ability to repurpose pretrained GANs via finetuning and transfer learning will become increasingly valuable.
We focus our work on GAN finetuning methods that can improve the quality of generated outputs and increase performance across a variety of metrics. While early chapters focus on unconditional generation, we later transition to conditional GANs, which ultimately become the main focus of our final chapter. We briefly describe the contents of each chapter.

1.1 Summary of Contributions

In Chapter 2, we identify Alpha Model Domination (AMD), a common obstacle to obtaining diverse, high-quality outputs in Multiple Choice Learning (MCL). We introduce several loss functions that consistently avoid AMD-related training failures [6]. Thereafter we focus our work on GAN inference time methods, which naturally support the diversity goals of MCL. Chapter 3 introduces CoachGAN [5], an inference time algorithm to improve GAN outputs. CoachGAN requires no modification of model architectures and works with any differentiable generator loss. Chapter 4 introduces a puzzle-based algorithm for finetuning generator models. Our approach encourages generators to produce more coherent and visually appealing results, which we substantiate with quantitative user study analyses. In Chapter 5, we address a common inference heuristic known as the truncation trick. We introduce a light-weight, feed-forward network as an improved version of the truncation trick. For Chapter 6, we focus on StyleGAN2 image projection, where we reduce the time needed to embed arbitrary images into the latent-noise space from 20 minutes to 2 seconds. In our conclusion, we reiterate our contributions and discuss implications of our work.
References


Chapter 2

Alpha Model Domination

Abstract

Multiple Choice Learning (MCL) algorithms produce several possible predictions so that an ‘oracle’ user with unmodelled biases can select the most preferred output prediction. Recent research has demonstrated that ensembles that implicitly or explicitly maximize diversity can significantly improve performance across a variety of MCL tasks. We identify a significant shortcoming and potential learning obstacle of the recent Stochastic Multiple Choice Learning (sMCL) algorithm, which we define as Alpha Model Domination (AMD). Using the CIFAR-10 and ImageNet image classification datasets, as well as the CamVid semantic segmentation dataset, we demonstrate the frequency and impact of AMD on MCL learning. We introduce and evaluate several novel sMCL loss functions that consistently avoid AMD, while increasingly trivially the required computation time.

2.1 Introduction

The practice of team member specialization arises naturally in numerous contexts such as sports, music, and wildlife. In these situations, individual member diversity can affect group success or even survival. We use this metaphor to motivate the machine learning task of $M$-best ensemble prediction. Unlike traditional machine learning, which maps a single input $X_i$ to a single output $y_i$, $M$-best prediction maps $X_i$ to $M$ plausible outputs. One of the earliest works in $M$-best prediction produced multiple possible solutions to the shortest path problem ([86]). Researchers have since demonstrated the value of $M$-best ensemble predictions in various fields such as machine translation ([49]), computational biology, and computer vision.

Potential use cases for $M$-best predictions arise naturally in computer vision tasks with ambiguous notions of correctness, for example, image captioning, inpainting, or image denoising. Each of these applications might involve end users with output preferences or biases not accounted for during training. These situations could allow a $M$-best ensemble to provide multiple answers from which the user would select an option. For instance, a user might have unspecified preferences in sentence style or length for an image captioning task. Given $M$ possible captions for input image $i$, the user could simply select the caption that best suits his or her subjective preferences.

In order for users to benefit from $M$ answers, ensembles must generate a diverse set of output predictions. This introduces the question of how ensembles should measure and encourage model diversity. As noted by [34], many ensemble training approaches implicitly encourage model diversity (e.g. using randomized weight initialization). More recent methods ([56, 95, 172]) explicitly enforce diversity using an additional diversity term in the loss function.

By strictly enforcing diversity, current training methods may produce diverse but useless members of ensembles. Furthermore, existing methods often require the sequential
training of multiple models. For complex deep learning tasks that often require days or weeks to train a single model, sequential training can intractably increase total training time. Research has introduced parallelized methods ([5, 90, 91]) to help avoid protracted training. While these parallelized methods require additional memory and computation resources, they substantially reduce the total time for training and prediction.

Because of this, we focus our work on Stochastic Multiple Choice Learning (sMCL), a recent and effective parallelized, diverse $M$-best ensemble training approach ([91]). For each training instance of sMCL, only the model with lowest error receives a parameter update. This leads to a common training issue, which we refer to as Alpha-Model-Domination (AMD).
In AMD, a single model receives the majority of updates and other ensemble models receive too few parameter updates to be useful. Our solution to AMD involves randomly relaxing the constraint that only the best performing model receives parameter updates.

Our contributions are as follows:

1. We identify and define Alpha-Model-Domination (AMD), an obstacle to consistent sMCL training.
2. We provide a number of efficient solutions to overcome AMD when training sMCL ensembles.

In section 2.2 we provide background in training diverse ensembles and sMCL. Section 2.3 defines AMD and analyze the its frequency in sMCL training using image classification and segmentation datasets. Section 2.3.2 introduces our novel loss terms that effectively overcome AMD. We describe our experimental setup and analyze results in section 2.4. We conclude with the implications of our research and outline future research directions in section 2.5.

2.2 Related Work

Diversity encouraging $M$-best algorithms stem from diversity methods in single-output ensemble prediction. For single-output regression, ensembles can reduce variance and thus overall error by combining the outputs of several models that yield independent prediction errors. According to [101], several independent regression models can reduce both ensemble variance and MSE when dealing with small-sized data sets. This finding helped cement the implicit rule that greater ensemble diversity reduces generalization error.

Later research in single output regression sought to increase diversity and reduce MSE using a bias-ambiguity decomposition. This decomposition,

$$E = \hat{E} - \hat{A}$$ (2.1)
where $\hat{E}$ is the summed individual model error and $\hat{A}$ is the ensemble ambiguity (i.e. a diversity measure), shows that an increase in second loss term can reduce overall ensemble loss ([22]). [81] likewise noted that a larger ambiguity term means that a greater value will be subtracted from the ensemble error. However, greater ensemble ambiguity implies increased errors for individual models. In other words, the ambiguity term simultaneously subtracts from the loss, $\hat{E}$, while increasing its starting value.

More recent ensemble training methods do not rely on loss decompositions, but hand-designed loss terms to encourage model diversity ([40, 82]). While these new diversity loss terms have provided notable improvements, they provide an inexact measure of diversity and thus may harm, rather than help, overall ensemble performance. One example of this is Negative Correlation Learning (NCL) ([95]), which encourages diversity by adding an output correlation penalty term to the loss function of each model in the ensemble. In standard NCL, the loss for an arbitrary model $j$ is defined as

$$L_j = \frac{1}{2}[F_j(n) - y(n))^2 - \lambda(F_j(n) - F(n))^2]$$  \hspace{1cm} (2.2)

where $F_i$ denotes the output of model $i$ on instance $n$, $y(n)$ is the instance target value, $F(n)$ is the ensemble output, and $\lambda \in [0, 1]$ is the weighting of the correlation penalty loss term. Although NCL appeared to show improvements by increasing ensemble diversity, [27] later demonstrated that NCL with $\lambda = 1$ mathematically reduces to training a single model training without regularization.

Various work in both regression and classification ensembles have introduced NCL-inspired, hand-designed loss functions to encourage ensemble diversity. For instance, [103] uses Jenson Shannon Divergence (JSD) to increase diversity in classification ensembles. JSD is a symmetric divergence measure

$$E_i = e_i - \lambda JSD(y_i || \hat{y})$$  \hspace{1cm} (2.3)
where $e_i$ is a differentiable loss, $\lambda \in [0,1]$ is the weight of the diversity term, $y_i$ is the output for model $i$, and $\hat{y}$ is the average output for all models $j \neq i$. Both NCL and JSD methods, however, require careful tuning of $\lambda$ in order to boost ensemble performance. Furthermore, the benefits of JSD diversity decrease as the label space size increases.

### 2.2.1 Multiple Choice Learning

In tasks such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), models often fail to disambiguate closely related classes. Because of this, researchers often report both traditional accuracy and top-$k$ accuracy. Top-$k$ accuracy compares an instance label to the $k$ classes with the highest predicted probabilities. In other words, an instance is correct if a model or ensemble contains the true label among its top-$k$ predictions.

Echoing top-$k$ metrics, Multiple Choice Learning (MCL) ([55]) uses a related metric for ensembles known as Oracle Accuracy (OA). We define OA for a single instance as

$$1 - \min_i l(y_i, \hat{y})$$

where $y_i$ is the output for model $i$, $\hat{y}$ is the true label, and $l$ is an arbitrary loss computed between $y_i$ and $\hat{y}$. Unlike top-$k$ measures, however, OA represents the selection of an ‘oracle’ user. This oracle views the $M$-best solutions generated by the ensemble and selects the best one, perhaps based upon a subjective criteria. In such scenarios, MCL models aim to create both useful and diverse predictions to meet the unspecified biases or preferences of a arbitrary end user. Examples of computer vision use cases include interactive image inpainting, object recognition, and semantic segmentation. These multiple predictions can provide useful solutions in interactive computer vision tasks.

The precursors of MCL involved single models and focused simply on generating the $M$ most likely predictions, rather than $M$ diverse methods. For instance, [43] uses a LP relaxation to sequential obtain the best $M$ solutions for a probabilistic graphical model.
After obtaining the best solution, the LP method generates the second best solution, and by induction provably yields the $M$ most likely solutions. [11] fused message passing algorithms with the M-best LP relaxation to reduce training time by multiple orders of magnitude.

Subsequent $M$-best ensemble methods included the additional objective of diversity when generating $M$-best solutions. Initial MCL approaches relied on dissimilarity functions to find diverse sets of probable solutions ([12]). [55] approached this problem by training ensembles with a two-step training process: Models train to convergence, after which the algorithm partitions the data by assigning each instance to the ensemble model that yielded the lowest prediction error during training. Models again train to convergence using their assigned subsets, thus becoming 'specialists' on different portions of the input domain. This resembles various single-output ensemble prediction methods ([5, 61]), which aim to train diverse models using separate generalist and specialist training steps.

Despite avoiding the need for computationally expensive diversity loss terms, [55] requires time consuming data resampling and model retraining. For massive data sets and deep learning models with millions of training parameters, repeated training and data partitioning can intractably extend learning time. [91] provides a more efficient, parallelizable training method, Stochastic Multiple Choice Learning (sMCL), which avoids both costly diversity loss terms and time-consuming retraining.

While training, sMCL calculates the loss, $l$, of each deep neural network model in the ensemble for every data instance. However, only the model with the smallest error receives a parameter update. This means that the loss for model $j$ is

$$loss_j = l(p_j) \mathbb{1}(j = \arg\min_x \{l(p_1), \ldots, l(p_M)\})$$

where $M$ is the number of ensemble members, and $\mathbb{1}$ is an indicator function that returns 1 when $l(p_j)$ yields the lowest loss for the current instance. Despite the elegance and efficiency of sMCL, closer analyses of the algorithm reveals a subtle yet problematic weakness. To
understand this drawback of sMCL, consider an ensemble of neural networks with randomly initialized weights. If an arbitrary model, $j$, starts out in a slightly better configuration than all other ensemble members for training batch $b$, model $j$ will receive the majority, if not all, of the batch parameter updates. For batch $b + 1$, model $j$ will have an improved parameter configuration and likely receive most of the parameter updates once again. This process continues throughout training at the expense of other models, producing weak, unhelpful, or even harmful ensemble members.

### 2.3 Alpha Model Domination

We refer to this problem as ‘alpha model domination’ (AMD). AMD occurs when a single model dominates the learning process at the expense of weaker models. We could similarly describe this scenario as ‘model starvation,’ where a model fails to perform better on a sufficient portion of the dataset. Consequently, stronger models in the ensemble will continue to improve and receive weight updates. Weaker models, however, will remain largely untrained and will generally harm performance at test time. We explicitly define AMD as one or more models receiving $\frac{N}{M}$ fewer updates than another model in the ensemble, where $N$ is the total number of instances and $M$ is number of ensemble members. As an example, given $N = 10,000$ instances and $M = 4$, an ensemble exhibits AMD if the difference between the models with the most and fewest parameter updates exceeds $\frac{10,000}{4} = 2,500$. Rather than compute all pairwise differences between model parameter updates, we can identify AMD by examining the greatest and smallest parameter update counts for the ensemble.

In order to demonstrate the effects of AMD, we ran experiments on the CIFAR-10 datasets with ensembles consisting of 4 models. Using the CIFAR-10-Quick convolutional neural network configuration provided with Caffe ([69]), we trained CIFAR-10 ensembles for 3,500 iterations. All ensembles trained with a batch size of 350, weight decay of 0.004, and a fixed learning rate of 0.001, similar to [91]. We ran each ensemble 3 times and recorded
the oracle accuracy, total number of parameter updates, and final test set accuracy for each model. Tables 2.3.1 and 2.3.2 show the results from these experiments.

Table 2.3.1: CIFAR-10 sMCL Accuracy and Oracle Accuracy

<table>
<thead>
<tr>
<th>Model</th>
<th>Run 1 Accuracy</th>
<th>Run 2 Accuracy</th>
<th>Run 3 Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3185</td>
<td>0.3637</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>0.1925</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>0.1877</td>
<td>0.3283</td>
<td>0.2743</td>
</tr>
<tr>
<td>4</td>
<td>0.2756</td>
<td>0.1</td>
<td>0.4013</td>
</tr>
<tr>
<td>OA</td>
<td>0.9232</td>
<td>0.8722</td>
<td>0.8755</td>
</tr>
</tbody>
</table>

Table 2.3.2: CIFAR-10 sMCL Number of Parameter Updates

<table>
<thead>
<tr>
<th>Model</th>
<th>Run 1 Updates</th>
<th>Run 2 Updates</th>
<th>Run 3 Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>431,201</td>
<td>532,987</td>
<td><strong>119,988</strong></td>
</tr>
<tr>
<td>2</td>
<td>238,428</td>
<td><strong>119,415</strong></td>
<td><strong>119,735</strong></td>
</tr>
<tr>
<td>3</td>
<td>238,385</td>
<td>418,400</td>
<td>355,312</td>
</tr>
<tr>
<td>4</td>
<td>282,336</td>
<td><strong>119,548</strong></td>
<td>595,315</td>
</tr>
</tbody>
</table>

Out of the three CIFAR-10 runs, two exhibited symptoms of AMD. Specifically, in Run 2 of the CIFAR-10 experiments, model 1 dominates at the expense of models 2 and 4, which never achieve better-than-random prediction accuracies. Run 3 on CIFAR-10 shows a similar situation where models 1 and 2 fail to learn useful information and model 4 receives more than 50% of all parameter updates. In addition to wasting valuable space and computation time, ensembles that experience AMD tend to yield lower OA scores. We note that for the CIFAR-10 experiments, runs 2 and 3 yielded OA scores nearly 5% lower than Run 1.

We also identified AMD in semantic segmentation experiments using the CamVid dataset [20], which contains 367 train and 233 test images with pixel-wise labelings using 11 semantic classes. Similar to the CIFAR-10 experiments, we trained several independent four member ensembles. Section 2.4.2 provides the full training details of these experiments. Figure 2.1.1 demonstrates the impact of AMD on a semantic segmentation sMCL ensemble.
While one of the ensemble members learns successfully to segment CamVid images, the remaining models learn to produce random or slightly better than random segmentations. This suggests that more complex computer vision tasks, for instance, semantic segmentation and image captioning, may experience more severe cases of AMD.

2.3.1 AMD Frequency

To better understand the frequency and impact of AMD, we conducted additional experiments using the CIFAR-10 and Imagenet classification datasets. For the CIFAR experiments, we used the same architectures and training details described in section 2.3. ImageNet experiments use the CaffeNet architecture and hyperparameter configuration included in the Caffe deep learning framework. For each data set, we train four-member ensembles from scratch, which we run for ten independent trials. Because of differences in data set difficulty, however, we train CIFAR-10 trials for 5000 iterations with batch sizes of 350 and ImageNet trials for 120,000 iterations with batch sizes of 256. This allows sufficient training time to identify AMD and demonstrate that AMD is a persistent problem that does not resolve itself with longer training time. Tables 2.3.4 and 2.3.5 show the final oracle accuracy and whether or not the ensemble experienced AMD in each of these experiments.

Table 2.3.3: Results for 10 trials using the CIFAR-10 and ImageNet datasets. Average final accuracy and proportion of models experiencing AMD is shown.

<table>
<thead>
<tr>
<th>Table 2.3.4: CIFAR 10 Runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
</tr>
<tr>
<td>0.9036</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2.3.5: ImageNet Runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
</tr>
<tr>
<td>0.4732</td>
</tr>
</tbody>
</table>

The CIFAR-10 results (Table 2.3.4) demonstrate that AMD consistently appears during training and detracts from OA. Specifically, the two non-AMD trials averaged 0.918
OA compared to 0.899 OA for AMD trials. The Imagenet results (Table 2.3.5) similarly confirm the frequent appearance of AMD. For these trials, however, the impact of AMD on OA is more subtle. At first glance AMD appears to positively impact OA. By examining how many ensemble members experienced model starvation in each trial, we deduced that the worst cases of AMD actually led to higher OA.

However, the improved OA in these trials is a natural result of the brief training times. Given only 120000 * 256 potential parameter updates, four-member ensembles that do not experience AMD will receive just ~7,680,000 updates per model. An AMD ensemble that allows a single model to receive on almost all of the 30,720,000 parameter updates will seemingly outperform the non-AMD ensembles. With additional training time, however, alpha-models eventually saturate in learning capacity and yield only marginal increases in OA. As we demonstrate in Section 2.4, non-AMD ensembles eventually surpass and produce better OA with the same amount of training time.

2.3.2 AMD Prevention

By initializing all ensemble models with weights from a pretrained model, ensembles can generally avoid AMD troubles in MCL training and prediction. While this approach works effectively for simple image classification tasks, for which numerous pretrained models appear online, other tasks and domains may not have pretrained models available. A simple workaround might train a single model for a new task and then initialize ensemble members with the weights of the newly-trained model.

However, this approach incurs the exact problem which sMCL-like methods attempt to avoid; namely, this method requires additional time to train separate generalist and specialist models. For difficult tasks that already require days to weeks of training time, the generalist-specialist approach can frustratingly compound the total amount of training time. Similar to [91], we aim to allow simultaneous training of $M$ diverse ensemble members from scratch. With recent advances in GPU power and parallel optimization, such an approach
is not only feasible, but provides an attractive means to quickly train ensembles to handle \(M\)-best problems effectively.

We now introduce three extensions to MCL-based methods that help avoid AMD during training. Each of these extensions introduces a variation of randomly awarding parameter updates to ensemble members. We detail the specifics of each method in the following subsections, after which we empirically evaluate and analyze these approaches in section 2.4. We expect each of these methods to both avoid AMD and boost average OA for the ensemble.

### 2.3.3 Random Selection

Under this setting, the ensemble awards a parameter update to the top performing ensemble member for a particular instance, much like sMCL. However, Random Selection (RS) also awards a second randomly selected model \(b\%\) of the time, where \(b \in \{10, 20, 30\}\). This random parameter update award allows models that may begin in poor parameter starting spaces to avoid model starvation and become useful ensemble members. Ideally, models that start poorly move toward favorable parameter spaces and eventually receive weight updates as a top-performing model. By encouraging all models in this fashion, each ensemble member will more likely learn to specialize on a subset of the input domain. At worst, models may overlap in their specializations. However, RS prevents the wasteful training that occurs when insufficiently trained, alpha-dominated models simply output random predictions. For our experiments, we compared two variations of this method: Updating all non-top performing models (RSA), and only updating a single randomly selected model (RSS).

### 2.3.4 Random Annealing

The Random Annealing (RA) sMCL extension nearly matches the RS approach. However, we start all ensembles with \(b = 50\%\) and linearly anneal the random award to 0 over the
training period. Mathematically, we write

\[ b = \frac{T - t}{T} \times 50 \quad (2.6) \]

where \( T \) is the total number of training epochs and \( t \) is the current epoch. RA aims to provide more opportunities for all models to receive parameter updates during the early stages of training to escape local minima. Toward the end of training, however, the probability of non-top models receiving a parameter update for a specific instance decreases significantly. This encourages greater input space specialization toward the end of training. Similar to RS, we experiment with three variations of RA. Specifically, we award a parameter update to the second best performing model with \( b \) probability (RAS), or we select a model to award randomly (RAR), or we award all \( M - 1 \) non-top performing models (RAA).

### 2.3.5 Stochastic Softmax

The Stochastic Softmax (SS) prevention method is a novel neural network loss layer that selects a model to update according to a multinomial distribution. To generate this probability distribution, SS accepts as input the softmax output distributions from all \( M \) ensemble models. After extracting the target class probability from each model, SS transforms these \( M \) predictions to a probability distribution that sums to 1.0 using a softmax function with simulated annealing:

\[ \frac{e^{p_j/\tau}}{\sum_{i=0}^{M} e^{p_i/\tau}} \quad (2.7) \]

where \( p_j \) is model \( j \)'s output for the target class, and \( \tau \) is the current annealing temperature. We exponentially decay \( \tau \) as

\[ \tau_t = \begin{cases} \tau_{\text{start}}(.999)^t & \tau_{t-1} > \tau_{\text{min}} \\ \tau_{\text{min}} & \text{otherwise} \end{cases} \quad (2.8) \]
where $\tau_{\text{start}}$ is the initial temperature for the softmax equation, $\tau_{\text{min}}$ is the minimal allowed temperature, and $t$ is the current training iteration. We set the minimum value of $\tau$ at either 1 or .1 in our experiments to avoid underflow or division by zero.

2.4 Experiments

In order to demonstrate the effectiveness of our proposed AMD prevention methods, we evaluate the RS, RA, and SS extensions on two different computer vision problems, image recognition and semantic segmentation.

2.4.1 CIFAR-10

This section describes the training details and results of our CIFAR-10 experiments. We used the machine learning framework Caffe to train and test all of our ensembles variations. Ensembles consisting of 4 models trained from scratch for 5,000 epochs for 3 independent trials. We ran all experiments on a single machine with an Intel Core i7-4770 processor, 32GB of RAM, and an Nvidia GTX 660 GPU. Table 2.4.1 reports the results from these experiments.

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>OA</th>
</tr>
</thead>
<tbody>
<tr>
<td>sMCL</td>
<td>0.8903</td>
</tr>
<tr>
<td>RSS b=0.1</td>
<td>0.929</td>
</tr>
<tr>
<td>RSA b=0.1</td>
<td>0.9031</td>
</tr>
<tr>
<td>RAS</td>
<td>0.9336</td>
</tr>
<tr>
<td>RAR</td>
<td><strong>0.9453</strong></td>
</tr>
<tr>
<td>RAA</td>
<td>0.9256</td>
</tr>
<tr>
<td>SS $\tau \in [3,1]$</td>
<td>0.9319</td>
</tr>
<tr>
<td>SS $\tau \in [3,0.1]$</td>
<td><strong>0.9389</strong></td>
</tr>
</tbody>
</table>

We note that most of the RSS or RSA trials yielded only marginally improved OA compared to standard sMCL. However, both methods successfully avoided AMD in all trials.
Out of the RA variations, RAR appears to provide the most consistent improvement over the baseline approach. SS methods also yield consistent improvements over sMCL.

### 2.4.2 Image Segmentation

We further evaluated the best performing sMCL extension, RAR, on the CamVid data set. Individual models mirrored the network architecture and parameter settings presented in [9]. For these experiments, we used the Caffe extension presented in [90], which employs the Message Passing Interface standard to train massive networks across multiple machines and GPUs. This extension allowed us to train ensembles of four Segnet models spread across four machines and four Nvidia K40 GPUs.

Tables 2.4.3 and 2.4.4 show the average OA test set results for 5 trials that each trained over three days for 40,000 iterations through the training data. Although sMCL on average achieves better OA, RAR ensembles produce noticeably smaller variance across runs. One reason for this is that the sMCL experiments exhibited mild-to-severe cases of AMD. In other words, 1-3 ensemble members in each sMCL run received fewer than 25% of the total available parameter updates. In fact, most starved models received fewer than 1% of the potential parameter updates. RAR ensembles, on the other hand, distributed parameter updates more uniformly.

The results of the ImageNet and CamVid experiments provide a number of useful insights for practitioners approaching MCL-related computer vision tasks. We have demonstrated that sMCL often wastes considerable computational resources and can occasionally yield inferior results. Our extensions, on the other hand, provide much more consistent methods to achieve similar or better results. For computer vision tasks that require concurrent training of multiple models over an extended period, our AMD prevention methods provide reliable, inexpensive means to reduce variance in ensemble performance.
Table 2.4.2: Averaged results for 5 trials of 4-member sMCL and RAR ensembles on the CamVid dataset.

<table>
<thead>
<tr>
<th>Table 2.4.3: sMCL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OA</td>
<td>Variance</td>
</tr>
<tr>
<td>0.5665</td>
<td>1.327 $\times 10^{-4}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2.4.4: RAR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OA</td>
<td>Variance</td>
</tr>
<tr>
<td>0.5508</td>
<td>1.1636 $\times 10^{-6}$</td>
</tr>
</tbody>
</table>

2.5 Conclusion

We note that this work does not aim to disparage sMCL or its many variants. When a pretrained model is available, sMCL provides an efficient method to improve OA scores. Nevertheless, as machine learning continues to spread to new areas with increasing speed, relevant models with pretrained weights are less likely to be available. Thus, future training scenarios underscore the usefulness of a rapid, parallelizable process for producing $M$-member diverse ensembles. Each of our novel approaches avoid AMD with only a marginal increase in computational cost and running time. Out of the proposed approaches, RAR and SS not only avoid AMD but occasionally yield higher OA. We thus submit that our methods represent effective extensions to sMCL that can improve performance and help avoid wasted resources and unnecessary ensemble retraining.

Future work may consider AMD from a more theoretical viewpoint. This type of analysis could provide precise estimates of the frequency of AMD in MCL-like training methods. Future work may also investigate OA over longer training periods. We conjecture that the greatest advantages of using RAR and other sMCL extensions may arise when ensembles train for several weeks. In this type of scenario, the alpha models in sMCL would likely saturate and struggle to improve individual or ensemble accuracy. AMD avoidance methods, however, provide steadier training schedules and would more likely allow improvement to continue throughout training and avoid model saturation.
### 2.6 Appendix

**Table 2.6.1: Expanded CIFAR 10 Oracle Accuracy at 5000 Epochs**

<table>
<thead>
<tr>
<th>Method</th>
<th>TRIAL 1</th>
<th>TRIAL 2</th>
<th>TRIAL 3</th>
<th>AVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>sMCL</td>
<td>0.9232</td>
<td>0.8722</td>
<td>0.8755</td>
<td>0.8903</td>
</tr>
<tr>
<td>RSS b=0.1</td>
<td>0.9305</td>
<td>0.9283</td>
<td>0.9281</td>
<td>0.929</td>
</tr>
<tr>
<td>RSS b=0.2</td>
<td>0.9172</td>
<td>0.9151</td>
<td>0.917</td>
<td>0.9164</td>
</tr>
<tr>
<td>RSS b=0.3</td>
<td>0.9037</td>
<td>0.9039</td>
<td>0.9073</td>
<td>0.905</td>
</tr>
<tr>
<td>RSA b=0.1</td>
<td>0.9045</td>
<td>0.9006</td>
<td>0.9041</td>
<td>0.9031</td>
</tr>
<tr>
<td>RSA b=0.2</td>
<td>0.8982</td>
<td>0.896</td>
<td>0.8979</td>
<td>0.8974</td>
</tr>
<tr>
<td>RSA b=0.3</td>
<td>0.897</td>
<td>0.9008</td>
<td>0.8975</td>
<td>0.8984</td>
</tr>
<tr>
<td>RAS</td>
<td>0.9313</td>
<td>0.9337</td>
<td>0.9359</td>
<td>0.9336</td>
</tr>
<tr>
<td>RAR</td>
<td>0.945</td>
<td>0.9431</td>
<td>0.9478</td>
<td>0.9453</td>
</tr>
<tr>
<td>RAA</td>
<td>0.928</td>
<td>0.9222</td>
<td>0.9266</td>
<td>0.9256</td>
</tr>
<tr>
<td>SS $\tau \in [3,1]$</td>
<td>0.9315</td>
<td>0.9319</td>
<td>0.9322</td>
<td>0.9319</td>
</tr>
<tr>
<td>SS $\tau \in [3,0.1]$</td>
<td><strong>0.9425</strong></td>
<td><strong>0.9374</strong></td>
<td><strong>0.9369</strong></td>
<td><strong>0.9389</strong></td>
</tr>
</tbody>
</table>
Table 2.6.2: Expanded results for CIFAR-10 and ImageNet datasets. Each trial reports the final accuracy and if AMD occurred during training (+).

Table 2.6.3: CIFAR 10 Runs

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8936</td>
<td>+</td>
</tr>
<tr>
<td>0.9047</td>
<td>+</td>
</tr>
<tr>
<td>0.9211</td>
<td>-</td>
</tr>
<tr>
<td>0.9124</td>
<td>+</td>
</tr>
<tr>
<td>0.8814</td>
<td>+</td>
</tr>
<tr>
<td>0.9111</td>
<td>+</td>
</tr>
<tr>
<td>0.8734</td>
<td>+</td>
</tr>
<tr>
<td>0.9046</td>
<td>+</td>
</tr>
<tr>
<td>0.9182</td>
<td>+</td>
</tr>
<tr>
<td>0.9152</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2.6.4: ImageNet Runs

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4508</td>
<td>+</td>
</tr>
<tr>
<td>0.5251</td>
<td>+</td>
</tr>
<tr>
<td>0.4534</td>
<td>+</td>
</tr>
<tr>
<td>0.5042</td>
<td>+</td>
</tr>
<tr>
<td>0.440647</td>
<td>+</td>
</tr>
<tr>
<td>0.401611</td>
<td>-</td>
</tr>
<tr>
<td>0.517467</td>
<td>+</td>
</tr>
<tr>
<td>0.464269</td>
<td>+</td>
</tr>
<tr>
<td>0.501661</td>
<td>+</td>
</tr>
<tr>
<td>x</td>
<td>?</td>
</tr>
</tbody>
</table>
References


Chapter 3

CoachGAN

Abstract

CoachGAN provides an inference time method to improve outputs from GAN generator models. Similar to creating adversarial examples to fool neural network classifiers, CoachGAN exploits gradient information, in this case from a pretrained discriminator model. Unlike the generating adversarial examples, which uses gradient descent to alter outputs directly, CoachGAN alters the inputs of generator models. This allows for output enhancements at test time without additional model training. CoachGAN adapts easily to existing algorithms and is architecture agnostic. In addition to qualitative samples, we quantitatively show that CoachGAN improves IS and FID scores on a variety of GAN architectures and tasks.

3.1 Introduction

Generative Adversarial Networks (GANs) can produce impressive results across a variety of tasks. The traditional GAN setup involves generator and discriminator models in a mini-max training scenario trying to optimize opposing loss functions. It is common to discard the discriminator after training and use only the generator model to produce novel synthetic outputs. We introduce an efficient post-training algorithm, CoachGAN, that exploits information in the discriminator at inference time to generate more realistic outputs.

At inference time, CoachGAN depends on a well-trained discriminator model that can accurately classify images as real or fake. This provides the generator with otherwise unavailable feedback on output quality at inference time. Rather than update the weights of the generator, which might reduce future generation quality, CoachGAN alters the input to improve realism. Metaphorically, CoachGAN takes on the role of an advisor that provides feedback to the generator at inference time.

Previous work \[26, 37\] explore input-centric methods to generate adversarial examples to improve the robustness of an auxiliary neural network classifier. CoachGAN instead uses information synthesized by a discriminator model to refine generator inputs to be more realistic. Unlike previous methods, CoachGAN does not require training additional models \[37\] or access to real training data during inference \[26, 37\]. Figure 3.1.1 demonstrates the gradual output refinement of CoachGAN using a DCGAN \[24\] generator and discriminator trained on the CelebA dataset.

CoachGAN provides an efficient inference time method that requires no modification of the generator and discriminator architectures. In addition, CoachGAN easily adapts to any GAN architecture with differentiable models and loss functions. We demonstrate this in our experiments with several unique GAN architectures and a variety of datasets. This work provides the following contributions:
Figure 3.1.1: Applying CoachGAN to images generated by a GAN trained on the CelebA dataset. Each row shows the transition from original image (left) to final output (right).

- The CoachGAN algorithm, which uses a pretrained discriminator to improve generator outputs at inference time.

- A wide variety of empirical results that demonstrate the effectiveness and adaptability of CoachGAN.

- A quantitative comparison of CoachGAN and non-CoachGAN outputs using Inception Score and Fréchet Inception Distance.

In Section 3.2 we briefly review relevant work. In Section 3.3, we introduce and discuss the CoachGAN algorithm. Section 3.4 outlines experiments and discusses qualitative and quantitative results. Section 3.5 summarizes this work and examines implications and paths for future work.

### 3.2 Related work

To the best of our knowledge, no previous work exploits information in the discriminator at test time for the sake of improving generator output. However, previous work has explored
generator input adjustments to favor a more natural look for adversarial examples, which are then used to increase the overall robustness of a classifier model.

Early adversarial methods ([16, 19, 23]) use backpropagation to add gradient-based noise and create ‘adversarial’ examples that convincingly fool neural network classifiers. Such approaches employ algorithms such as the Fast Gradient Sign Method [10] to exploit the linear behavior of neural networks when dealing with high dimensional inputs [20]. Similarly, [21] synthesizes the preferred inputs of a classification model for each class by performing activation maximization on the output neurons. While these various methods can effectively fool neural network classifiers, generated images often contain unrealistic curves, distortions, and color blending.

Several works map an adversarially altered training instance, $\hat{x}$, to a latent space vector, $z^*$, such that $\hat{x} \approx G(z^*)$. For instance, [37] uses an inverter network, $I_γ$, to map $\hat{x}$ to $z^*$. Alternatively, $z^*$ can be found by optimization to minimize differences between $G(z^*)$ and $\hat{x}$ [26]. By projecting adversarial images onto the range of $G$, these methods can remove unrealistic blur and artifacts to produce more natural-looking images. While these approaches produce more natural-looking adversaries, [37] requires training an additional inversion model, and [26] samples several $z$ and attempts to minimize a non-convex optimization task.

Several works introduce image editing tools that manipulate low-level latent spaces that approximate the natural image manifold. For example, users suggest facial feature changes [6] or color and structure edits [38] in pixel space and a model performs the gradient updates in the latent space. While these methods generally result in more coherent and visually pleasing images, they require training models that predict the latent vector, $z^*$, that most closely matches a user’s edits. A technique similar to CoachGAN is used for image inpainting [35], but some components of the algorithm (e.g. pixel distance-weighting to corrupted image regions) do not generalize to other GAN tasks.

The Deep Image Prior (DIP) method [30] asserts that a randomly initialized neural network is an effective image prior as the result of low-level, structural information that exists
implicitly in the network architecture. However, DIP requires an iterative, computationally-
expensive optimization of an entire randomly-initialized neural network for every single input
instance. CoachGAN, on the other hand, can improve large batches of outputs simultaneously.
Furthermore, CoachGAN naturally extends to non-image domains, while DIP is exclusively
suited to the image domain.

Other methods, such as the GLO framework [3], similarly perform optimization in the
latent space. While GLO gives a training-time approach to optimize an embedding space
using Laplacian pyramid and $\ell_2$ losses, CoachGAN provides an inference time method that
works with arbitrary differentiable losses. Also, despite encouraging results on the CelebA
dataset, GLO performs poorly compared to GANs on larger datasets such as LSUN [3].

The introspective generative modeling approach generates textures by performing
gradient accent in pixel-space over a series of $T = 20$ trained CNN classifiers [17]. In contrast,
CoachGAN uses a single discriminator and performs gradient descent in latent space to
improve generator outputs at inference time. An advisor analogy is also used in [34] to
describe training two distinct generative models using MCMC sampling. Despite a similar
metaphor, CoachGAN differs in purpose and approach. Our method is architecture-agnostic
and runs at inference time.

3.3 Method

CoachGAN provides a post-training, modular approach to improve the outputs of a generator.
Given input $z$ and pretrained $G$ and $D$ models with frozen weights, CoachGAN improves
output realism by gradually altering $z$ using backpropagation and gradient descent. The
input $z$ is not limited to latent space vectors, but can take the form of any continuous input
space. CoachGAN uses the same differentiable loss, $L_G$, used during training for $G$. In this
case, however, CoachGAN backpropagates $L_G$ through $D$ and $G$ and performs a gradient
descent update only on \( z \). Thus at timestep \( t \) CoachGAN computes the loss

\[
L_t = L_G(G(z_t))
\]  

(3.1)

and \( z \) receives a gradient update according to the chosen optimization method. Under basic stochastic gradient descent optimization, \( z_{t+1} \) would update as

\[
z_{t+1} = z_t - \eta \frac{\partial L_G}{\partial z}
\]  

(3.2)

where \( \eta \) is the learning rate. At time \( t + 1 \), CoachGAN computes the loss as

\[
L_{t+1} = L_G(G(z_{t+1}))
\]  

(3.3)

This CoachGAN optimization process repeats for a user-specified number of iterations, \( \kappa \).

As a specific example, consider the original minimax GAN objective function:

\[
\min_G \max_D \log(D(x)) + \log(1 - D(G(z)))
\]  

(3.4)

Traditionally, an optimizer for \( G \) minimizes \( L_G = \log(1 - D(G(z))) \), or it maximizes \( \log(D(G(z))) \), which helps avoid vanishing gradient problems [9]. CoachGAN uses the same \( L_G \) as in training, but does not compute gradients with respect to \( \theta_G \), the weights of \( G \). Instead, CoachGAN computes gradients with respect to the original input, \( z \). In this work, we use the improved Wasserstein GAN loss [11], where CoachGAN attempts to minimize \( L_G = -D(G(z)) \). In the following subsubsection, we discuss the CoachGAN loss surface and path taken by gradient descent.

### 3.3.1 Theory

The work of [28] shows that the latent space manifolds of deep neural networks approximate zero curvature. This suggests that movement in latent space \( z \) resembles geodesics, which
minimize the distance between output points [31]. This idea is commonly used in GAN spherical interpolation methods [29, 33], which produce more realistic output transitions than linear interpolations between outputs. From this viewpoint, we can consider CoachGAN as a partial optimization of latent sample \( z \) in an \( \mathbb{R}^{z|} \)-dimensional manifold defined by \( L_G \). Similar to geodesics, small movements in the \( z \) latent space can quickly produce realistic transitions in the output.

Like heuristic activation and weight regularization techniques, CoachGAN does not provably guarantee improved output realism. However, empirical results suggest that CoachGAN tends to improve generated outputs.

### 3.3.2 Intuition

To demonstrate the behavior of CoachGAN, we conduct a simple experiment using the pretrained DCGAN \( G \) and \( D \) that generated the results shown in Figure 3.1.1. We sample a single 100-dimensional \( z \) vector from a spherical Gaussian. Since visualizing the effects of CoachGAN in \( \mathbb{R}^{100} \) space is infeasible, we perform the following simplifications: We hold all \( z_k \) constant, where \( k \in [3, 100] \), and only allow CoachGAN to change \( z_1 \) and \( z_2 \). For reference, we plot the outputs of \(-D(G(z))\) when varying \( z_1 \) and \( z_2 \) from -4 to 4 by increments of 0.4. This provides an intuitive illustration of CoachGAN’s loss landscape and behavior throughout optimization.

We initialized CoachGAN with \( z_1 = z_2 = 0.0 \) and allowed the algorithm to run for 200 iterations using an Adam optimizer with a learning rate of 0.01. Figure 3.3.1 plots the path traveled by CoachGAN as well as sample outputs. Subfigures 3.3.1(a-c) and the associated marks on the graph correspond with CoachGAN outputs at iterations 1, 100, and 200, respectively. The realism of the images increase with more iterations, though the results are limited when only optimizes 2 out of 100 dimensions. Figure 3.3.1(d) shows the final output at 200 iterations when CoachGAN can optimize all \( z_k \), not just \( z_1 \) and \( z_2 \).
Figure 3.3.1: Starting at $z_1 = z_2 = 0.0$, CoachGAN uses an optimizer (Adam used here) to follow the direction in $z$-space that minimizes that value of $-D(G(z))$. To visualize the effects of our method, we only allow CoachGAN to update $z_1$ and $z_2$. Images (a-c) and the associated marks on the graph correspond with CoachGAN outputs at iteration 1, 200, and 400, respectively. Image (d) shows the output generated when CoachGAN optimizes the full $z$ vector.

In later experiments, we use a smaller learning rate and fewer coaching iterations to prevent significant changes in the identity of the original output (e.g. Subfigure 3.3.1(d)). We employed the high $\eta$ and $\kappa$ values solely to demonstrate the movement through the loss surface.

3.3.2.1 Basins of Attraction

One consideration in evaluating the usefulness of CoachGAN is whether or not CoachGAN decreases the number of possible outputs from $G$. In other words, does CoachGAN guide
$z$ to a limited number of basins in the input space? Also, does CoachGAN simply push $z$ toward previous $z$ values encountered during training?

To answer these questions, we conducted a simple experiment using DCGAN $G$ and $D$ models and the MNIST dataset. We train the models for 10 epochs on the training set and record all 600,000 $z_t$ used in training. After training, we sample an additional 5,000 random $z_i$ and perform CoachGAN for 10 iterations with a learning rate of 0.01, which yields the coached input, $z_{ic}$. To test whether similar but unique $z$ converge to the same basin, we also perform CoachGAN on $z'_i = z_i + \mathcal{N}(0, \epsilon)$, where $\epsilon = 0.001$. This finds the coached input $z'_{ic}$.

To quantify the relationship between $z_i$ and $z_{ic}$, as well as $z_{ic}$ and $z'_{ic}$, we compute a number of statistics. First, we calculate distance of $z_c$ to the the nearest $z_t$ in $z$-space. To calculate this value we average the L2-norm of the vector difference between $z_i$ and the nearest $z_t$ from the training set.

$$\text{Nearest-Training-Z-Distance} = \frac{1}{n} \sum_i \min_t ||z_i - z_t||$$ \hspace{1cm} (3.5)

Next we calculate the distance in pixel space between $G(z_i)$ and $G(z'_{ic})$ as the L2-Norm between the outputs, which we average across all 5,000 samples.

$$\text{Pixel-Distance} = \frac{1}{n} \sum_i ||G(z_i) - G(z'_{ic})||$$ \hspace{1cm} (3.6)

Finally, we calculate the average $z$-space distance between $z_{ic}$ and $z'_{ic}$.

$$\text{Z-Distance} = \frac{1}{n} \sum_i ||z_{ic} - z'_{ic}||$$ \hspace{1cm} (3.7)

This measures the similarity of the $z$ vector coaching paths given only a small amount of Gaussian noise differentiating the inputs.

The top-left plot in Figure 3.3.2 shows both that $z_i$ and $z'_{ic}$ yield measurably different output images, and CoachGAN does not simply push $z_i$ toward $z$ encountered during training.
Figure 3.3.2: 2D histograms comparing the relationships of pixel distance, z-space distance, and nearest training z distance metrics. Even with a difference of just $\mathcal{N}(0,0.001)$, $z_i$ and $z'_i$ often produce visually distinct outputs. Furthermore, CoachGAN does not push $z$ vectors toward basins of attraction around $z_t$.

The top-right plot shows that $z_i$ and $z'_i$ end up at distinct $z$ after undergoing CoachGAN refinement. The bottom plot confirms the positive correlation between pixel-distance and z-space distance.

Figure 3.3.3 shows sample outputs from this experiment. Our results demonstrate that CoachGAN does not push $z$ vectors into large basins of attraction, or significantly reduce the number of possible outputs. Even when $z_i$ and $z_{ic}$ differ by just $\mathcal{N}(0,0.001)$, the resultant outputs often show visual distinctions. This suggests that CoachGAN makes dimension specific updates based on the overall state of an embedding vector. A tiny amount of noise added to an input vector can alter which dimensions CoachGAN updates at inference time.

### 3.3.2.2 Discussion

Dense, high-dimensional target domain spaces makes the generation of realistic outputs for all $z$ highly improbable. Training algorithms like Wasserstein GAN [1] and Improved
Figure 3.3.3: Output comparison of the original and coached samples, $G(z_i)$ and $G(z_{ic})$, to $G(z'_i)$ and $G(z'_{ic})$, where $z'_{ic} = z_i + \mathcal{N}(0, 0.001)$. Even with a small amount of random noise (which does not noticeably alter the original output), visual differences appear in the resultant coached outputs.

Figure 3.3.4: Left to right: transition from original image to final CoachGAN output. The leftmost image shows a faint, partially generated set of glasses. By pushing the source input vector toward the closest element-wise modes, CoachGAN allows $G$ to generate a clear set of glasses.

Wasserstein GAN [11] encourage smoother interpolations in the output space and generally improve convergence. However, these methods still struggle to achieve full mode-coverage due to factors like insufficient model capacity or a poorly enforced Lipschitz gradient constraint [27].

CoachGAN provides a method to push portions of the input into domain areas that lead to more confidently realistic outputs. As an example, consider the output shown in Figure 3.3.4. The leftmost image shows the faint outline of a pair of glasses around the man’s eyes. CoachGAN performs element-wise adjustments on $z$ using gradient descent, which results in a more pronounced pair of glasses in the final right image. Effectively,
CoachGAN encourages small coordinated steps in the continuous domain space to generate a more believable final output.

### 3.3.3 Optimizing output directly

For sake of comparison, we experimented with optimizing the output of $G$ directly, rather than altering the original input. This removes the need to backpropagate through $G$ and slightly speeds up the CoachGAN process. Unfortunately, this approach does not produce the same high quality results as the input-altering CoachGAN algorithm. Figure 3.3.5 shows an example output of CoachGAN when optimizing the outputs for the CelebA dataset. Instead of increasing the realism of outputs, this method simply adds adversarial-like noise [10].

### 3.4 Experiments

CoachGAN does not assume a particular optimization algorithm, but we use Adam for our experiments. We found a basic tradeoff between the chosen number of iterations, $\kappa$, and the optimizer learning rate, $\eta$. For example, in Figure 3.4.1, we show the output refinement for a single image with $\eta \in \{0.002, 0.004, 0.01\}$ (rows) and $\kappa \in \{5, 10, 20, 40\}$ (columns). The
diagonal from top right to bottom left reveals that the various settings produced similar output images. For faster convergence, CoachGAN can use a larger $\eta$ and smaller $\kappa$. In general, however, we found the best results with smaller $\eta$ such as 0.001 and $\kappa$ between 50 and 100. This appears to hold true for a variety of algorithms and datasets, which we discuss in subsequent sections.

CoachGAN adapts easily to various training algorithms and provides noticeable improvements to generated outputs. We evaluate CoachGAN on the original unconditional GAN architecture [9] using the CelebA and LSUN bedroom datasets. Adding CoachGAN required minimal code alteration and often led to substantial qualitative improvements, as we demonstrate in the following subsubsections.

We emphasize that CoachGAN improves the majority of GAN outputs. Under certain circumstances, such as poorly trained models or blurry training images, we observed that CoachGAN magnifies existing noise in outputs. Additionally, if the initial output, $G(z)$, already appears realistic, CoachGAN does not directly improve realism, but adjusts image characteristics to match those most favored by $D$. We provide specific examples of this $D$ favoritism in our results. We also provide randomly sampled CoachGAN outputs and quantitative evaluation to demonstrate the general behavior of CoachGAN.

3.4.1 Unconditional GAN

We first present results for the basic unconditional GAN algorithm. We employ the DCGAN [24] $G$ and $D$ models and the WGAN-GP [11] training algorithm for these experiments.

3.4.1.1 CelebA

For the CelebA experiments, we trained $G$ and $D$ for 40 epochs using the same parameters as [24]: A learning rate of 0.0002, batch size of 64, and the Adam optimizer with $\beta_1 = 0.5$ and $\beta_2 = 0.999$. We center cropped the training data and resized images to 64x64 for faster processing.
As demonstrated in Figures 3.1.1, 3.3.1, and 3.3.4, CoachGAN can provide remarkable improvements to low quality outputs from $G$. These results illustrate the effectiveness of CoachGAN in performing a wide variety of realism enhancements at inference time.

When the output images of $G$ already appear realistic, CoachGAN pushes outputs to resemble the modes of the dataset, as captured by $D$ during training. For instance, Figure 3.4.2 reveals that $D$ favors centered, face-forward images. This is not unexpected, as most images in the CelebA dataset possess these characteristics. Other modes of $D$ that we empirically observed include brightening images, removing bangs, and reducing baldness.

### 3.4.1.2 Nearest neighbor

Similar to Section 3.3.2.1, we verify that CoachGAN does not push generated outputs toward existing training samples. In order to test this, we perform a 5-Nearest Neighbor search on the training set. Rather than use a distance metric in the output space, we adopt the method of [7], which computes the distance in feature space using the activations of several layers of a pretrained VGG-19 network. Figure 3.4.3 displays the nearest neighbor results for both an
CoachGAN reveals learned rotation biases of $D$. Both the man and woman are centered with eyes facing forward.

Figure 3.4.3: 5-Nearest Neighbors in the training set for the original generated output (top-left) and the CoachGAN refined output (bottom-left).

original output sample and the CoachGAN refined output. The results show that CoachGAN does not push outputs toward existing training set samples. Rather, CoachGAN greedily adjusts inputs in $Z$ space to generate outputs that better fool $D$.

3.4.2 LSUN

Using the same DCGAN and WGAN-GP setup as the CelebA experiments, we further explored the effects of CoachGAN on downsampled 64x64 LSUN bedroom images. Figure 3.4.4 shows the CoachGAN transformations for 7 randomly sampled $z$ vectors. These outputs demonstrate that CoachGAN is not limited to highly regular datasets, such as the centered and aligned CelebA face images. For the generated LSUN outputs, CoachGAN clarifies the
edges of bedding and walls, removes blurring, and even adds details to windows and reflective surfaces.

3.4.3 Progressive Growing of GANs

An increasing number of research groups provide access to their pretrained GAN weights online. This means that CoachGAN can readily be used with a variety of publically available state-of-the-art methods. To demonstrate the ease with which CoachGAN adapts to other models, we applied CoachGAN to the Progressive Growing of GANs (PGGAN) [14] inference method. Expanding upon earlier pyramid-based [8] or multi-step GAN training algorithms [36], PGGAN generates outputs of increasingly higher resolution - ultimately producing a $1024 \times 1024$ output images.
Figure 3.4.5: Left: Original images generated with the Progressive Growing of GANs model [14]. Right: The resultant images after applying CoachGAN.
With only a few lines of additional code, we successfully augmented PGGAN inference with CoachGAN. Figure 3.4.5 shows samples comparing original (left) and CoachGAN outputs (right). In the top image, CoachGAN largely removes the color blob covering the man’s head. As for the middle image, CoachGAN improves the hair texture and removes artifacts from the forehead. In the bottom image, CoachGAN seals a gap in the woman’s neck and completes a partially created earring.

3.4.4 StyleGAN

The recent StyleGAN model [15] builds upon style transfer methods [13] to yield an unprecedented level of high-definition sample quality as well as a better disentangled feature space. Besides introducing and using the larger 70,000-image FFHQ dataset (compared to 30,000 images in CelebA-HQ), StyleGAN relies on the truncation trick (TT) [5], which reduces the latent sampling space in order to improve image quality. While the resampling approach of TT performs a similar function as CoachGAN, TT reduces image variation – especially when using low truncation values. In contrast, CoachGAN can yield noticeable improvements with just tiny changes to the original latent vector, as demonstrated in Section 3.3.2.1.

TT also does not scale well to arbitrary generator architectures. For instance, [5] notes that orthogonal regularization of the weights of each layer of the generator is needed for TT to work effectively in their BigGAN method. CoachGAN, on the other hand, works with any differentiable GAN architecture and does not require layer modification or access to the generator weights. We note that TT relies on random resampling of the input latent vector, whereas CoachGAN uses gradient descent to more intelligently search for a ‘improved’ latent vector. We expect that CoachGAN can yield improvements even after applying TT. To test this claim, we compare FID scores of StyleGAN using TT with a threshold of 0.7 (as used in [15]) and TT + CoachGAN in Section 3.4.8.
3.4.5 Conditional GAN

3.4.6 BigGAN

Google’s class conditional BigGAN [5] significantly improved state-of-the-art FID and IS scores for condition GAN generation. Although the official online repository provides only pretrained generator weights for several image resolutions, the primary BigGAN author released a PyTorch version of BigGAN, which includes pretrained G and D models. Because this unofficial model trained on just ImageNet (rather than expanded dataset used in the paper), the model does not attain the state-of-the-art results reported in [5]. However, we still observe improvements in IS and FID score, as we demonstrate in the next section.

3.4.7 Quantitative evaluation

Inception Score (IS) [11] remains one of the most popular and widely adopted GAN evaluation metrics. Using a representative image sample from generator, $G$, IS produces class label distributions using a pretrained Inception v3 classification neural network. IS then calculates the Kullback-Leibler divergence between the class distribution of each generated output, $p(y|G(z_i))$, and the average label distribution of all samples, $p(y)$. The exponentiated expectation of these KL-divergences yields the final IS score. We write this as

$$IS(G) = exp( \mathbb{E}_{G(z_i)} KL( p(y|G(z_i)) \ || \ p(y) ) ) \quad (3.8)$$

The work introducing IS, [25], states that IS tends to correlate well with human opinion of image realism.

Since IS calculates an entropy distribution over 1,000 ImageNet classes, we do not measure IS on the single-class datasets such as CelebA and LSUN-bedroom. We do, however, measure IS on a DCGAN trained on CIFAR-10 for 200 epochs and a BigGAN model trained for 100 epochs. Due to limited model capacity (DCGAN) and reduced training time and data (BigGAN), we do not observe state-of-the-art IS values in these experiments. Rather,
we hypothesize that CoachGAN will lead to a relative increase in IS for each of the GAN experiments.

Following the recommendation of [2], we generate 50,000 samples for calculating IS. We report the initial IS and IS after applying CoachGAN with $\kappa = 10$ and $\eta = 0.01$ for CIFAR and $\kappa = 1$ and $\eta = 0.01$ for BigGAN. The results in Table 3.4.1 show that CoachGAN improves the baseline IS score.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>IS-PRE</th>
<th>IS-POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>4.79</td>
<td>5.00</td>
</tr>
<tr>
<td>BigGAN</td>
<td>60.30</td>
<td>60.48</td>
</tr>
</tbody>
</table>

Table 3.4.1: Inception Score calculated before (PRE) and after CoachGAN (POST), where higher scores are better.

3.4.8 Fréchet inception distance

Although IS remains a popular metric for GAN evaluation, its use is limited for datasets that do not share classes with ImageNet. In fact, an increasing number of theoretical and empirical analyses [2, 4, 22] demonstrate that IS does not measure intra-class diversity and fails to detect training set memorization. Additionally, IS relies on an Inception model pretrained on the 1000-label ImageNet dataset, which may not be appropriate for non-ImageNet GAN evaluation tasks (e.g. GANs trained on data with image statistics and label distributions that differ noticeably from ImageNet).

Because of this, we evaluate CoachGAN using the Fréchet Inception Distance (FID) [12], which is calculated based on the activations of a 2048-dimension pooling layer in the Inception v3 network. Using real data samples, $X$, and generated samples, $G(Z)$, FID is calculated as

$$FID = ||\mu_X - \mu_{G(Z)}||^2 + \text{Tr}(\Sigma_X + \Sigma_{G(Z)} - 2(\Sigma_X \Sigma_{G(Z)})^{\frac{1}{2}})$$  \hspace{1cm} (3.9)$$

We also calculate FID using coached $Z'$ values.
Using pretrained CIFAR-10, LSUN-bedroom, and CelebA DCGAN models, we use $\kappa = 10$ and $\eta = 0.01$ to compare FID before and after CoachGAN. We calculate these scores using FID statistics computed over the entire training datasets and 50,000 generated or coached images. CoachGAN produces better (lower) FID scores for two out of three of the DCGAN models (see Table 3.4.2).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>PRE</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>CelebA</td>
<td>17.95</td>
<td>16.43</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>35.29</td>
<td>34.38</td>
</tr>
<tr>
<td>LSUN</td>
<td>24.22</td>
<td>24.34</td>
</tr>
<tr>
<td>PGGAN</td>
<td>54.36</td>
<td>54.13</td>
</tr>
<tr>
<td>StyleGAN</td>
<td>16.74</td>
<td>16.55</td>
</tr>
<tr>
<td>BigGAN</td>
<td>40.35</td>
<td>40.20</td>
</tr>
</tbody>
</table>

Table 3.4.2: Frechet Inception Distance calculated with $z$ samples before (PRE) and after applying CoachGAN (POST). CoachGAN improves FID (lower is better) for both CelebA and CIFAR-10 pretrained DCGAN models.

We also calculate pre- and post-CoachGAN FID scores for PGGAN, StyleGAN, and BigGAN using $\kappa = 1$ and $\eta = 0.01$ (we observed similar results for various tradeoffs of $\eta$ and $\kappa$). Because Celeb-HQ only contains 30,000 training images, it is not ideal for calculating FID scores with 50,000 samples PGGAN images. The FFHQ dataset, which is an expanded version of the Celeb-HQ dataset from [15], contains 70,000 images. We use FFHQ as the ground truth dataset for scoring both PGGAN and StyleGAN. Table 3.4.2 reports the FID results for these experiments. All three models yield improved FID scores when using CoachGAN.

3.4.9 Runtime

While CoachGAN works best with small learning rates and more iterations, we also observe improvements for single-step coaching with larger learning rates (as demonstrated in the PGGAN, StyleGAN, and BigGAN experiments). Single-step coaching can improve results with a moderate increase in running time. To quantify this statement, we compare generation
times with and without CoachGAN for DCGAN, PGGAN and BigGAN in Table 3.4.3. We use an NVIDIA Tesla P100 GPU for these experiments.

<table>
<thead>
<tr>
<th></th>
<th>Normal</th>
<th>CoachGAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCGAN</td>
<td>0.0009073</td>
<td>0.00535</td>
</tr>
<tr>
<td>PGGAN</td>
<td>0.02202</td>
<td>0.0229</td>
</tr>
<tr>
<td>BigGAN</td>
<td>0.06157</td>
<td>0.18806</td>
</tr>
</tbody>
</table>

Table 3.4.3: Generation runtime (in seconds) with and without CoachGAN.

Although using CoachGAN with BigGAN (the model with the greatest complexity) roughly triples inference time, the total time is still under 0.2 seconds. In our experiments, coaching 50,000 samples with a batch-size of 16 took less than an hour on a single GPU. Larger batch sizes and a more powerful GPU could further reduce the time needed for CoachGAN.

### 3.5 Conclusion

CoachGAN provides a modular, effective approach to improve generator outputs at inference time. We have empirically demonstrated qualitative and quantitative improvements using CoachGAN with a variety of datasets. We also demonstrated the ease of applying CoachGAN to different GAN architectures with distinct loss functions (e.g. DCGAN, PGGAN, StyleGAN, and BigGAN). Regarding current applications, there is an increasing interest in using GANs for image editing and synthesis [6, 18, 32, 38]. Such applications require high quality results and the potential for user control. CoachGAN can both refine poor quality results and allow users to select a precise output from sequences of images generated by CoachGAN’s SGD steps.

Future work will consider task-specific constraints to preserve desired features. For instance, CoachGAN could restrict changes that alter characteristics such as gender, hair color, or pose when improving outputs of the CelebA dataset. This could prove useful when
applying CoachGAN to video sequence generation, where certain visual features must stay constant from frame to frame.
References


Chapter 4

PuzzleGAN

Abstract

Previous work has employed puzzle-solving to improve image segmentation, object recognition, and video classification. To date, no existing work has used puzzle-based losses to improve GAN output quality. We introduce PuzzleGAN, an auxiliary end-to-end differentiable method for finetuning generator models. Using the Progressive Growing of GANs (1024x1024) and BigGAN (128x128) models, we demonstrate that PuzzleGAN can improve FID score by nearly 5 points with just 100k iterations of finetuning. We perform extensive FID and IS quantitative comparisons of PuzzleGAN against baseline finetuning approaches. Our puzzler finetunings demonstrate competitive and often superior performance compared to baseline methods. We observe unique characteristics in qualitative image samples and neuron saliency map visualizations of Puzzler networks. We conclude that puzzle-based losses provide a unique and helpful gradient signal for GAN finetuning.
Figure 4.1.1: Comparison of generated outputs from pretrained (left) and finetuned with PuzzleGAN (right) BigGAN-deep models. The PuzzleGAN output exhibits a more natural focus on the image center, while reducing background clutter and increasing continuity of table edges.

4.1 Introduction

Despite improvements in Generative Adversarial Network (GAN) training via stagewise training \[29\], novel architectures \[30\], and massive datasets \[7\], GANs still struggle with producing consistently realistic outputs. While GANs excel at creating realistic image textures locally, they struggle to create straight lines and globally coherent images without visual artifacts. Figure 4.1.1 demonstrates this issue with images sampled from the state-of-the-art BigGAN-deep [7] generator.

We introduce a jigsaw-puzzle-based method, PuzzleGAN, for finetuning arbitrary GAN architectures. PuzzleGAN splits the generator output into a number of square pieces divisible by 4. We choose base 4 to easily divide images of sizes 128x128 and 1024x1024 into \(k\) equally sized squares. After splitting the output image into \(k\) squares, we randomly shuffle and feed the \(k\) pieces into a neural network that predicts the correct ordering.

While previous approaches have introduced a variety of different losses to improve GAN performance \[3, 5, 24, 36, 39, 46, 57, 63\], none to the best of our knowledge explore a puzzle auxiliary task to improve global image fidelity. Existing puzzle solving literature provides feature learning methods, such as the relative ordering task of two randomly selected image patches in \[18\]. While effective, this approach does not involve a complete ordering of patches. Other puzzle-related works focus on specific problems \[31, 35, 59\] or employ a
discrete approximation for the task of puzzle permutation recovery [53]. In this work, we introduce a novel end-to-end differentiable puzzle-loss, which allows for easy adaptation to arbitrary GANs.

We show that finetuning with puzzle loss feedback can help the generator learn to produce more realistic, connectable pieces. By optimizing for permutation recoverability, the generator produces images with improved spatial correlations [10] and salient cross-boundary relationships [19]. Using pretrained Progressive Growing of GANs [29] and BigGAN [7] models, we evaluate PuzzleGAN finetuning for unconditional and conditional GAN image generation, respectively.

Our contributions include:

1. DCGAN-based and ResNet-based auxiliary PuzzleGAN models, along with an end-to-end differentiable Puzzle-Loss for GAN finetuning.
2. Novel variation of SS-GAN [12] for finetuning, which does not require training from scratch.
3. Extensive quantitative evaluation and qualitative comparisons of image samples.
4. Saliency map network visualizations to inspect learned features of Puzzle models.

The rest of our paper is outlined as follows: We briefly review related work in Section 4.2. Section 4.3 formally introduces PuzzleGAN with associated Gamemaker and Puzzler networks. In Sections 4.4 and 4.5, we outline experiments and analyze results, respectively. Section 4.6 visualizes and discusses Puzzler neuron saliency maps. Section 4.7 summarizes our conclusions.

4.2 Related Work

Many previous works have improved upon the original GAN formulation introduced in [23]. Some of these methods improve gradient flow through the use of alternate loss functions [3, 5, 24, 36] or weight normalization techniques [39]. A more recent approach introduces an
attention mechanism to enable the learning of long-range dependencies in image generation models [63]. The BigGAN model [7] builds upon [63] and achieves impressive results by leveraging a massive dataset with extended training time and compute power.

Prior work has successfully demonstrated the ability to disentangle features in latent spaces of generator input [50]. This ability enables [32] to encode and enforce a facial-keypoint geometric prior in the input. Some methods [14, 55] use image heatmaps of facial keypoints to enforce facial geometric priors in outputs. [13] incorporates a human-pose geometric prior by training discriminators to distinguish between real and fake poses. While these methods improve the structural integrity of GAN outputs, they require supervised learning and task-specific geometry losses, which limits their generalization.

Despite improved training techniques and attention mechanisms to encourage global image synchronization, SOTA models still suffer from artifacts and obvious aberrations, as demonstrated in Figure 4.1.1. Auxiliary differentiable losses offer an attractive means to improve image coherence in existing GAN methods. Before introducing PuzzleGAN, we briefly review self-supervised feature learning and puzzle-based learning tasks.

### 4.2.1 Self-supervised learning

Previous work has leveraged various auxiliary losses for self-supervised learning. Some methods involve predictions such as the relative location of an image patch [18, 40] or classes assigned during unsupervised clustering [11]. Other prediction methods include the angle prediction of rotated images [21], colorization [33], or image inpainting [49]. While these and other works have leveraged surrogate tasks and losses to learn useful features for subsequent tasks, no existing GAN method to the best of our knowledge employs a puzzle-solving task during training or fine-tuning.
4.2.2 Puzzle-based losses

Previous works have introduced algorithms to tackle the NP-complete task \cite{17, 56} of solving a jigsaw puzzle \cite{9, 27}. While earlier works primarily focus on apictorial puzzle solving using combinatorial \cite{22, 60} or boundary matching methods \cite{2, 58}, more recent work approaches pictorial puzzle-solving tasks from a computer-vision perspective \cite{9}. For instance, \cite{19} leverages cross-boundary gradient distributions and \cite{44} computes image features to match up pieces based on edge similarity. \cite{44} and \cite{16} employ both shape and image features for solving jigsaw puzzle tasks.

While puzzle-solving naturally finds application in fields such as biology \cite{37}, archeology \cite{8}, and speech unscrambling \cite{66}, puzzle solving can also provide an effective self-supervised learning signal to improve model performance in various tasks. For instance, \cite{45} employs puzzle-solving as a pre-training step to learn semantic features for classification and object detection. As noted in 4.2.1, other auxiliary tasks such as image colorization \cite{33} or predicting 2D image rotation \cite{21} can serve as useful self-supervised means for semantic learning. We compare against this last rotation method in our experiments in Section 4.4.

\cite{31} demonstrated that puzzle-solving allows transfer learning to achieve state-of-the-art results on the PASCAL segmentation and classification tasks. The work of \cite{35} treats shuffled video images as a temporal puzzle in order to learn static image structure to improve performance in recognition tasks. \cite{59} finds that puzzle solving as a pretraining step improves results across a variety of tasks such as segmentation, detection, and classification. We conjecture that GAN output quality will similarly improve with puzzle-based training.

\cite{53} most closely mirrors this work. However, they parameterize their method as a discrete learning task of recovering a puzzle permutation matrix. With two neural networks, we cast the permutation matrix recovery task as an end-to-end differentiable learning problem. This removes the need for the loss approximation of \cite{53} and allows for easy integration with arbitrary GAN architectures. We now describe our approach.
4.3 Method

PuzzleGAN consists of two neural networks, a Gamemaker (GM) and a Puzzler (P). As mentioned in Section 4.1, we use 4 or 16 pieces because they easily divide into the dimensions of the images used in our experiments. We could extend our method to higher powers of 4, or even use shredded image pieces as in [34]. However, we focus on $4^1$ and $4^2$ pieces to reduce the training time for P models in our experiments.

GM uniformly samples one of the $k! - 1$ possible puzzle permutations (we exclude the trivial permutation of $1, 2, \ldots, k$) and then applies that permutation to the $\sqrt{k} \times \sqrt{k}$ sub-blocks of its input image. Formally, given an $n \times n$ color image, $I \in \mathbb{R}^{3 \times n \times n}$, GM outputs a shuffled image, $I_p \in \mathbb{R}^{3 \times n \times n}$, along with the corresponding label $\beta \in \mathbb{Z}^k$, where $1 \leq \beta_i \leq k$ and $\beta_i \neq \beta_j$ for $i \neq j$. The relationship between $I$ and $I_p$ is that the $j^{th}$ sub-block of $I$ is equal to the $i^{th}$ sub-block of $I_p$ if $\beta_i = j$.

Given $I_p$, P outputs a predicted ordering of the pieces, $\alpha \in \mathbb{R}^{k \times k}$. Unlike $\beta$, $\alpha$ consists of $k$ normalized probability vectors, each of length $k$. Each $\alpha_{kj}$ represents P’s predicted probability that the puzzle piece in position $k$ belongs in position $j$. This setup, which we diagram in Figure 4.3.1, allows for a straightforward negative-log-likelihood Puzzle Loss (PL):

$$PL = - \sum_k \sum_j \mathbb{1}(j = \beta_k) \log(\alpha_{kj} + \epsilon)$$ \hspace{1cm} (4.1) \\
$$+ (1 - \mathbb{1}(j = \beta_k)) \log(1 - \alpha_{kj} + \epsilon)$$ \hspace{1cm} (4.2)

where $\mathbb{1}$ is an indicator function that returns 1 if the inside expression evaluates as true (or 0 otherwise), and $\epsilon$ is a small value ($1e-7$ in our experiments) added to avoid undefined log input values.

By learning to unscramble shuffled pieces of images, P learns spatial correlations [10] and salient cross-boundary relationships [19]. We train P for several days on real image data prior to use in any GAN task. We note that GM, although structured as a neural
network, does not require training. GM consists of frozen 0/1 block matrices and serves as a straightforward, differentiable image-shuffling function.

P and GM are only added after G and D are trained in the normal fashion. Then we finetune G and D while keeping P weights fixed. If we jointly train P, G, and D, we encounter trivial pixel ‘cheats’ in generator outputs that correspond to target labels.

The majority of our experiments use P with model architectures based on the DCGAN discriminator [51]. The original DCGAN processes 64x64 images. To extend our model to larger image resolutions, we simply add additional DCGAN blocks of convolution, batch normalization [28], and ReLU [41] output activation. Each additional block doubles the latent space dimensions of the previous layer, while dividing in half the height and width of the input. We also explore using ResNet-18 (excluding the output layer) as a backbone for a puzzle output prediction layer. We discuss more details of this approach in Section 4.4.

### 4.3.1 Puzzler architectures and training

We first performed experiments to understand what input image sizes and Puzzler architectures would be best for our experiments in Section 4.4 with ImageNet [52] and CelebA-HQ [29]...
pretrained generators. Using 16-piece P models ($P_{16}$), we compare training progress on architectures that accept downscaled images of sizes 64x64 ($P_{16}^{64}$), 128x128 ($P_{16}^{128}$), and 256x256 ($P_{16}^{256}$) as input (see supplementary materials for full details). We trained each of these models on the ImageNet training set for 72 hours on a single Tesla P100 GPU. As mentioned in Section 4.3, this pretraining step does not involve G or D.

As Figure 4.3.2 demonstrates, $P_{16}^{64}$ and $P_{16}^{256}$ respectively process batches or reduce PL most quickly. However, $P_{16}^{128}$ provides the best balance between iteration speed and performance. For clarification, the horizontal axis in Figure 4.3.2 displays wall time up to 72 hours. We record PL after each completed training batch. We thus use downscaled 128x128 images as inputs to P in the rest of our experiments. We continued training $P_{16}$ for a total of 15 days on ImageNet. This further reduced PL down to $\sim 20$, which corresponds to correctly ordering 13/16 pieces for each image on average.

To investigate whether pretrained classifiers can serve as useful weight priors for P, we replaced the final layer of a pretrained 18-layer ResNet ImageNet classifier with a P output layer. We finetuned this model, $P_{16_{ResNet}}$, for 15 days. We compared finetuning all layers
and only finetuning the final output layer (i.e. using layers $l_1 \ldots l_{n-1}$ as feature extractors). However, models with only last-layer finetuning failed to perform better than random at puzzle-solving tasks. We thus use only the models in which all layers are finetuned using PL. While we observed similar PL reductions for both $P_{16}$ models across the 15-day period, $P_{16_{ResNet}}$ exhibited less oscillation and a steadier reduction in PL. However, $P_{16_{ResNet}}$ finished with a PL of 24 – slightly worse than its DCGAN counterpart. In addition to $P_{16}$ models, we trained DCGAN and ResNet $P_4$ models. Unlike $P_{16}$, which requires solving a more difficult puzzle task, $P_4$ models converged in roughly 10 hours.

For Celeb-HQ, we only trained DCGAN-based $P_4$ and $P_{16}$. We did not include ResNet models, since the classifier weights are trained on ImageNet, rather than face-based tasks. The more regular, eye-centered Celeb-HQ dataset allowed $P_4$ and $P_{16}$ to converge in under 72 hours.

### 4.3.2 Self-supervised GANs

We compare our models to the Self-Supervised GANs (SS-GAN) method introduced in [12]. We compare against this method because it achieves state-of-the-art results on unconditional ImageNet generation and has been shown to perform better than other self-supervised methods such as colorization [64], cross-channel prediction [65], and inpainting [18]. SS-GAN trains D and G with an additional rotation prediction loss, where images are rotated randomly at angle $R \in \{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$. During training, D tries to optimize the following loss:

$$
L_D = L_{GAN_D} - \beta \mathbb{E}_{r \sim R}[\log Q_D(R = r| x^r)],
$$

(4.3)

where $L_{GAN_D}$ is a usual discriminator loss (e.g. hinge loss), $\beta$ is a weight on the rotation loss, $x^r$ is a rotated real image, and $Q_D(R = r| x^r)$ is D’s predicted distribution over $x^r$’s rotation angles.
Although D trains on both real and fake samples for $L_{GAN_D}$, D’s rotation loss only processes real samples. D aims to learn how real images should look in order to appear realistic at any of the four possible angle rotations. This resembles our approach in training PuzzleGAN models, where P learns coherence features of real images that can be used to refine G’s weights during subsequent training.

In SS-GAN, G minimizes an $L_{GAN}$ loss as well as a rotation loss. The rotation loss is identical to Equation 4.3, excluding the unique rotation weight $\alpha$ and the input $G(z)$:

$$L_D = L_{GAN} - \alpha \mathbb{E}_{r \sim R}[\log Q_D(R = r|G(z))]$$  \hfill (4.4)

We use [12]’s recommended settings of $\alpha = 0.2$ and $\beta = 1.0$ in our experiments.

We note SS-GAN as presented in [12] trains models from scratch, whereas PuzzleGAN finetunes pretrained models. Because of this, we compare against a slightly modified SS-GAN, m-SSGAN, in which we pretrain D models with the usual GAN loss + rotation loss for 72 hours before finetuning G. We experimented with a brief ‘burn-in’ period with the extra SS-GAN output branch (linear layer + softmax) prior to finetuning G. However, this approach performed poorly and we opted for the 72-hours of D pretraining (see supplementary materials for illustration of this training process).

We note a design choice difference between PuzzleGAN and SS-GAN. SS-GAN attaches a second output branch to D, which learns both realism and rotation-related features. This approach offers the advantages of fewer models and feature sharing in D, but requires changing the architecture of D. We chose to use auxiliary GM and P models because puzzle-solving (especially with 16+ pieces) is more computationally intensive and may require additional parameters to learn features useful for puzzle solving. We explore the unusual and unique characteristics of P models with network visualization in Section 4.6. Using external puzzle-solving models improves modularity and allows for easy extension to new GAN architectures and tasks.
4.4 Experiments

In our experiments, we quantitatively evaluate finetuning variations for PGGAN and BigGAN. Following that, we explore randomly selected qualitative samples for PGGAN, BigGAN, as well as BigGAN-deep.

4.4.1 Finetuning setup

Datasets We use the Celeb-HQ [29] and ImageNet [52] datasets for unconditional and conditional GAN experiments, respectively.

Pretrained GANs For Celeb-HQ experiments, we use the Progressive Growing of GAN (PGGAN) architecture with the original pretrained G and D weights used in the paper [29]. For ImageNet experiments, we use a pretrained BigGAN [7] model published by one of the original paper authors. We note that this model is not the official TensorFlow version used in [7], but rather a PyTorch implementation released by the same authors as [7]. This version of BigGAN trained for 100k iterations on ImageNet to generate 128x128 images.

Puzzle variations We compare baseline model performance to finetuning with D, \(P_4\), \(P_{16}\), \(P_4 + P_{16}\), \(P_4 + D\), \(P_{16} + D\), \(P_4 + P_{16} + D\), and m-SSGAN. As noted in Section 4.3.2, we pretrain m-SSGAN models for 72 hours to adjust D to handle the additional rotational angle prediction branch. For experiments with multiple losses (e.g. \(P_4 + P_{16}\)), we calculate and backpropagate all losses for a training batch before updating G.

Metrics For BigGAN conditional generation experiments, we report Inception Score (IS) [24] and Fréchet Inception Distance (FID) [25], two commonly used metrics in GAN literature. Following [4]'s recommendations, we use 50,000 samples for all experiments. Since the 1000-class ImageNet label distribution used in IS does not extend naturally to single-class datasets [4, 6, 47] like CelebA-HQ, we only report FID for PGGAN experiments.

Learning rates We compared results across learning rates \(\eta \in \{1e-5, 1e-7, 1e-9\}\) for BigGAN models and \(\eta \in \{1e-6, 1e-7\}\) for PGGAN models. We evaluated a variety of other larger and smaller \(\eta\) values for PGGAN and BigGAN finetuning. However, we observed that
Table 4.5.1: FID Results for PuzzleGAN finetunings of PGGAN (lower is better).

<table>
<thead>
<tr>
<th></th>
<th>Orig</th>
<th>D</th>
<th>m-SSGAN</th>
<th>P4</th>
<th>P16</th>
<th>P4 + P16</th>
<th>P4 + D</th>
<th>P16 + D</th>
<th>P4 + P16 + D</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\eta = 1e-6)</td>
<td>-</td>
<td>40.69</td>
<td>22.07</td>
<td><strong>13.06</strong></td>
<td>16.98</td>
<td>15.68</td>
<td>16.39</td>
<td>18.30</td>
<td>16.72</td>
</tr>
<tr>
<td>(\eta = 1e-7)</td>
<td>14.10</td>
<td>13.83</td>
<td><strong>12.54</strong></td>
<td>12.84</td>
<td>13.22</td>
<td>13.06</td>
<td>13.08</td>
<td>13.37</td>
<td>13.15</td>
</tr>
</tbody>
</table>

\(\eta > 1e-6\) and \(\eta > 1e-5\) for PGGAN and BigGAN, respectively, led to exploding gradients. Similarly, \(\eta < 1e-7\) and \(\eta < 1e-9\) did not noticeably affect model weights during the finetuning period.

**Evaluation** We trained models for 100k iterations and evaluated snapshots every 25k iterations. Following the example of [30], we report the best result produced during finetuning.

### 4.5 Results

We first report FID and Amazon Mechanical Turk results for PGGAN unconditional generation experiments. We then report FID and IS for BigGAN conditional generation experiments.

#### 4.5.1 PGGAN

We report baseline FID results in Table 4.5.1 for pretrained, finetuning with D, and m-SSGAN. We also display the results of the various PuzzleGAN models.

For \(\eta = 1e-7\), all PuzzleGAN variations improved FID compared to pretrained and baseline finetuning with D. Although only \(P_4\) improved FID over the pretrained model FID for \(\eta = 1e-6\), the PuzzleGAN finetuning variations did not experience as great of a performance drop as finetuning with D alone or m-SSGAN. This suggests that P can act as a form of training stabilization. Of course, this stabilizing effect only holds true up to a certain point. As discussed in Section 4.4, learning rates greater than 1e-6 experienced gradient divergence and poor FID scores.

While PuzzleGAN outperformed pretrained and D finetuning, our m-SSGAN approach produced the lowest FID for \(\eta = 1e-7\). This demonstrates that pretraining the SS-GAN...
output branch of D and subsequently finetuning G works and can successfully improve FID score. We note that m-SSGAN finetuning performs better on PGGAN than BigGAN, as we demonstrate in Section 4.5.3.

Interestingly, the $P_{16}$ PuzzleGAN variations perform better than $P_{4}$ models in the BigGAN experiments, but $P_{4}$ outperforms $P_{16}$ in PGGAN experiments. This suggests a basic heuristic for selecting a PuzzleGAN architecture for finetuning: Use P models with fewer pieces for simpler, more regular datasets such a CelebA-HQ. Conversely, use P with more pieces for highly complex, diverse datasets such as ImageNet. We emphasize that actual behavior may differ on other datasets or models, and that this heuristic is only meant to provide a helpful starting point for future research.

4.5.2 Amazon Mechanical Turk

Amazon Mechanical Turk (AMT) [48] provides a convenient means to gather larger samples of human ratings on the realism of generated GAN outputs [1, 15]. We conduct an AMT experiment for which we generate 5,000 images from the 3 baseline methods and 6 puzzle variation using identical random seeds. To attain high quality results, we only allow workers who have successfully completed at least 5,000 other AMT tasks and have a 98% or higher approval rating. Following survey best practices [20, 38], we also include a trap question to validate human effort. We discard responses that do not correctly answer the validation question.

In each survey, responders view three sets of two randomly ordered outputs generated with the same random seed but different methods. For each comparison set, the worker drags a 0-100 value slider toward the most realistic-looking image (where 0 signifies that the left image looks most realistic, 100 signifies that the right image looks most realistic, and 50 indicates similar realism in both images). Table 4.5.2 reports the results for this experiment.

We highlight several interesting results from the AMT study. Unsurprising, D and m-SSGAN (which have much worse FID scores for $\eta = 1e-6$) received far lower ratings that
Table 4.5.2: AMT Results for PGGAN. Each table entry indicates the average slider value with the row image on the left and column image on the right. Values below 50 signify that humans prefer the row method (and vice versa).

<table>
<thead>
<tr>
<th></th>
<th>Orig</th>
<th>D</th>
<th>SSGAN</th>
<th>P₄</th>
<th>P₄+D</th>
<th>P₁₆</th>
<th>P₁₆+D</th>
<th>P₄+P₁₆</th>
<th>P₄P₁₆D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orig</td>
<td>-</td>
<td>10.8±1.4</td>
<td>10.5±1.6</td>
<td>50.5±2.2</td>
<td>40.6±2.7</td>
<td>56.5±2.7</td>
<td>53.8±2.9</td>
<td>58.2±2.3</td>
<td>58.6±2.5</td>
</tr>
<tr>
<td>D</td>
<td>89.2±1.4</td>
<td>-</td>
<td>55.5±3</td>
<td>89.5±1.5</td>
<td>89.8±1.3</td>
<td>91±1.3</td>
<td>91.5±1.2</td>
<td>91.2±1.3</td>
<td>90.2±1.3</td>
</tr>
<tr>
<td>SSGAN</td>
<td>89.5±1.6</td>
<td>44.5±3</td>
<td>-</td>
<td>90.3±1.5</td>
<td>89.5±1.6</td>
<td>91.7±1.3</td>
<td>92.1±1.4</td>
<td>91.6±1.2</td>
<td>91.6±1.4</td>
</tr>
<tr>
<td>P₄</td>
<td>49.5±2.2</td>
<td>10.5±1.5</td>
<td>9.67±1.5</td>
<td>-</td>
<td>41±2.7</td>
<td>56.3±2.6</td>
<td>57±2.5</td>
<td>55.8±2.5</td>
<td>56.5±2.7</td>
</tr>
<tr>
<td>P₄+D</td>
<td>59.4±2.7</td>
<td>10.2±1.3</td>
<td>10.5±1.6</td>
<td>59±2.7</td>
<td>-</td>
<td>63.4±2.7</td>
<td>63.6±2.5</td>
<td>64±2.6</td>
<td>63.2±2.5</td>
</tr>
<tr>
<td>P₁₆</td>
<td>43.5±2.7</td>
<td>9.04±1.3</td>
<td>8.28±1.3</td>
<td>43.7±2.6</td>
<td>36.6±2.7</td>
<td>-</td>
<td>47.8±2.4</td>
<td>48.5±2.2</td>
<td>49.4±2.3</td>
</tr>
<tr>
<td>P₁₆+D</td>
<td>46.2±2.9</td>
<td>8.54±1.2</td>
<td>7.88±1.4</td>
<td>43±2.5</td>
<td>36.4±2.5</td>
<td>52.2±2.4</td>
<td>-</td>
<td>47.6±2.5</td>
<td>48.4±2.2</td>
</tr>
<tr>
<td>P₄+P₁₆</td>
<td>41.8±2.3</td>
<td>8.77±1.3</td>
<td>8.37±1.2</td>
<td>44.2±2.5</td>
<td>36.2±2.6</td>
<td>51.5±2.2</td>
<td>52.4±2.5</td>
<td>-</td>
<td>49.5±2.4</td>
</tr>
<tr>
<td>P₄P₁₆D</td>
<td>41.4±2.5</td>
<td>9.84±1.3</td>
<td>8.4±1.4</td>
<td>43.5±2.7</td>
<td>36.8±2.5</td>
<td>50.6±2.3</td>
<td>51.6±2.2</td>
<td>50.5±2.4</td>
<td>-</td>
</tr>
</tbody>
</table>

other methods. P₁₆, P₄+P₁₆, and P₄+P₁₆+D have higher (worse) FID scores than the original pretrained PGGAN model. However, humans rated images from these finetuned models as significantly more realistic than the output from the original pretrained model. This finding mirrors our qualitative results in Section 4.5.4, which shows that these methods have the effect of removing aberrations and improving natural image quality.

### 4.5.3 BigGAN

In Table 4.5.3 (a), we report pretrained FID (lower is better) along with two baseline comparisons: Finetuning with D and m-SSGAN, which we introduced in Section 4.3.2. We report IS results (higher is better) for these same baselines in Table 4.5.3 (b).

**DCGAN Architecture** In the top half of Table 4.5.3 (a), we present the FID results for DCGAN-based Puzzlers. For \( \eta = 1e-7 \) and \( \eta = 1e-9 \), PuzzleGAN variations produced better FID than pretrained and baseline finetuning with D. In general, DCGAN finetuning setups that include \( P₁₆ \) performed best. As an example, \( P₁₆ + D \) yielded an FID of 35.91, which is a \( \sim 5 \) point improvement to the original pretrained BigGAN model after just 100,000 finetuning iterations.
Table 4.5.3: Left: FID results for BIGGAN (lower is better). Right: IS results for BIGGAN (higher is better). We score metrics using 50,000 samples with G models finetuned for 25k-100k iterations and report the best result across four model snapshots.

<table>
<thead>
<tr>
<th>Model</th>
<th>$\eta = 1e^{-7}$</th>
<th>$\eta = 1e^{-9}$</th>
<th>Model</th>
<th>$\eta = 1e^{-7}$</th>
<th>$\eta = 1e^{-9}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pretrained</td>
<td>-</td>
<td>40.35</td>
<td>Pretrained</td>
<td>-</td>
<td>28.96</td>
</tr>
<tr>
<td>$D$</td>
<td>38.46</td>
<td>38.36</td>
<td>$D$</td>
<td>36.83</td>
<td>37.28</td>
</tr>
<tr>
<td>$m$-$SSGAN$</td>
<td>40.08</td>
<td>38.27</td>
<td>$m$-$SSGAN$</td>
<td>33.78</td>
<td>37.44</td>
</tr>
<tr>
<td>DCGAN</td>
<td></td>
<td></td>
<td>DCGAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_4$</td>
<td>37.42</td>
<td>38.26</td>
<td>$P_4$</td>
<td>36.40</td>
<td>37.45</td>
</tr>
<tr>
<td>$P_{16}$</td>
<td>36.15</td>
<td>38.22</td>
<td>$P_{16}$</td>
<td>34.62</td>
<td>37.44</td>
</tr>
<tr>
<td>$P_4 + P_{16}$</td>
<td>36.24</td>
<td>38.33</td>
<td>$P_4 + P_{16}$</td>
<td>34.61</td>
<td>37.41</td>
</tr>
<tr>
<td>$P_4 + D$</td>
<td>37.40</td>
<td>38.26</td>
<td>$P_4 + D$</td>
<td>37.24</td>
<td>37.43</td>
</tr>
<tr>
<td>$P_{16} + D$</td>
<td>35.91</td>
<td>38.23</td>
<td>$P_{16} + D$</td>
<td>34.74</td>
<td>37.45</td>
</tr>
<tr>
<td>$P_4 + P_{16} + D$</td>
<td>35.97</td>
<td>38.33</td>
<td>$P_4 + P_{16} + D$</td>
<td>34.73</td>
<td>37.42</td>
</tr>
<tr>
<td>ResNet</td>
<td></td>
<td></td>
<td>ResNet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_4$</td>
<td>38.29</td>
<td>38.37</td>
<td>$P_4$</td>
<td>37.29</td>
<td>37.23</td>
</tr>
<tr>
<td>$P_{16}$</td>
<td>38.34</td>
<td>38.36</td>
<td>$P_{16}$</td>
<td>34.59</td>
<td>37.27</td>
</tr>
<tr>
<td>$P_4 + P_{16}$</td>
<td>38.42</td>
<td>38.37</td>
<td>$P_4 + P_{16}$</td>
<td>37.22</td>
<td>37.26</td>
</tr>
<tr>
<td>$P_4 + D$</td>
<td>38.37</td>
<td>38.37</td>
<td>$P_4 + D$</td>
<td>37.25</td>
<td>37.25</td>
</tr>
<tr>
<td>$P_{16} + D$</td>
<td>38.36</td>
<td>38.37</td>
<td>$P_{16} + D$</td>
<td>37.25</td>
<td>37.25</td>
</tr>
<tr>
<td>$P_4 + P_{16} + D$</td>
<td>38.40</td>
<td>38.37</td>
<td>$P_4 + P_{16} + D$</td>
<td>37.22</td>
<td>37.25</td>
</tr>
</tbody>
</table>
Figure 4.5.1: Comparison of DCGAN (top) and ResNet (bottom) BigGAN finetuning with $P_4 + P_{16} + D$.

We note that finetuning with only $P_{16}$ (i.e. no $D$ model) with $\eta = 1e-7$ yielded an FID of 36.15. This demonstrates that lightweight P models can provide a fast and computationally inexpensive way to improve G models. While finetuning with $D$ can further improve FID, the addition of $D$ almost doubles the required GPU memory and training time.

Table 4.5.3 (b) shows the IS results for DCGAN-based PuzzleGAN architectures. While all models for $\eta = 1e-7$ and $\eta = 1e-9$ improve over pretrained IS, we do not observe as much improvement as FID scores. Only $P_4 + D$ improves over standard finetuning with $D$. This may occur because IS, although correlated with image realism, is known to be sensitive to adversarial noise [4] or sharp image features [61].

**ResNet-18 Architecture** The FID results in Table 4.5.3 (a) demonstrate improvement over pretrained across the board for $\eta = 1e-7$ and $\eta = 1e-9$. However, the reductions in FID are not as noticeable as those of the DCGAN-based P. We note that ResNet-based P tend to be more resilient to changes in learning rates. We observed less model divergence in trials with $\eta > 1e-7$ for ResNet models (but all models with large $\eta$ performed worse than pretrained).

In the lower portion of Table 4.5.3 (b), we report IS results for the ResNet PuzzleGAN variations. ResNet experiments tend to produce stabler and higher IS scores than their
Figure 4.5.2: Image samples from PGGAN with $\eta = 1e-6$ to illustrate the effects of finetuning with different PuzzleGAN setups.

DCGAN-based counterparts. In Figure 4.5.1, we compare samples from DCGAN and ResNet finetunings with $P_4 + P_{16} + D$. The DCGAN outputs have less background noise and more detail on the main objects than ResNet. While this appears to improve FID for DCGAN models, it likely decreases IS, which can favor images with more sharply defined artifacts and noise diversity.

4.5.4 Qualitative results

To better intuit the quantitative results in Sections 4.5.1 and 4.5.3, we compare image samples from a subset of the various finetuned models. We include additional samples from all model variations in our supplementary materials. Because models both finetuned and sampled using identical random seeds, we can easily compare the differences in the generated outputs.

**PGGAN** In Figure 4.5.2, we display images from models finetuned for 100k iterations with $\eta = 1e-6$. $P_4$ tends to remove or smooth background noise – particularly in the corners of images. $P_{16}$, on the other hand, focuses on smoothing the inner pieces of the face. This
inner smoothing seems to reduce the complexity of the generated faces, which results in slightly worse FID scores for $P_{16}$ models.

We also sampled images from models finetuned using $\eta = 1e-7$. We include those results as well as samples from all $\eta = 1e-6$ model variations in our supplementary materials.

**BigGAN** Figure 4.5.3 shows images sampled from models finetuned with DCGAN-based P using $\eta = 1e-7$. As shown in Section 4.5.3, finetunings that include $P_{16}$ tend to produce the best results. The samples in Figure 4.5.3 support this claim. We note that both $P_4$ and $P_{16}$ tend to straighten surfaces and lines (see third column from left). Additionally, $P_{16}$ brightens and clarifies more complex details of images. This is particularly evident in the go-cart image of the right-most column of Figure 4.5.3. The outputs from top-to-bottom (ordered by improving FID) show progressively more detail and brightness added to the car frame.

To more clearly demonstrate the qualitative effects of PuzzleGAN finetuning, we present results with a higher resolution BigGAN-deep model in the next section. We do not
report quantitative results for this method (though we observed FID improvements in our trials), but present it primarily as a clear demonstration of PuzzleGAN finetuning.

**BigGAN-deep** Using a pretrained 256x256 BigGAN-deep model\(^1\), which does not include D weights, we finetuned the generator for 500,000 iterations with a batchsize of 10 using the DCGAN-based \(P_{16}\). In Figure 4.5.4, we compare samples of the original pretrained (left) and finetuned models (right). These samples show clearly the effect of training with \(P_{16}\): Fewer artifacts, smoother backgrounds, greater subject detail, and enhanced lighting. We include additional qualitative samples from BigGAN-deep in Figure 4.5.5.

### 4.6 Network visualization

To better understand the features learned by PuzzleGAN models, we compare network visualizations of the ResNet-18 \(P_{16}\) model to a ResNet-18 classification network pretrained on ImageNet. Neural network interpretation via visualization is an active qualitative research area with many different approaches. Earlier works in deep learning often use first-layer filter visualizations [26] to infer the visual features extracted by hidden layer neurons. Other approaches employ gradient descent to generate images that produce high activation values in a particular network neuron [42, 43]. However, these methods are known to produce unusual patterns and artifacts that do not convey clearly what a model has learned [54].

For our network visualizations, we leverage the work of [54], which uses a single back-propagation pass to generate image saliency maps for the most highly activated output neurons. Saliency maps tend to be more meaningful when generated from neurons in higher hidden layers, which show less sensitivity to input variation and greater class discrimination [62]. Because the ResNet classification and \(P_{16}\) models do not share the same output layer architecture, we use neurons from the final pooling layer of each network.

\(^1\)https://github.com/huggingface/pytorch-pretrained-BigGAN
Specifically, we pass batches of dog images from ImageNet through each network. We record the highest average neuron activations in the pooling layers along with the associated input images. Using simple backpropagation, we generate saliency maps that can be easily overlayed on the original input images. This allows us to observe the kinds of visual features relevant for higher-level semantic reasoning tasks in both networks.

In Figure 4.6.1, we display sampled images with overlayed saliency maps for some of the most activated pooling-layer neurons in the ResNet classifier. These saliency maps
Figure 4.5.5: Using identical $z$ and conditional classes, we show results after finetuning with DCGAN $P_{16}$ at 100k, 200k, 300k, 400k, and 500k iterations. 500k tends to produce ‘cartoon’-like outputs, which blur out the background add detail to the central object of the image. 300k-400k iterations provides semantic coherence (e.g., connecting separated limbs or paws) while better maintaining the original background details.

resemble those from existing literature [54, 62], and show relatively interpretable concepts such as fences, dog faces, or grass.

The saliency maps generated from the $P_{16}$ ResNet’s pooling layer differ noticeably from those of the classifier. We note that since the images which maximally activate neurons differ in each network, we cannot compare the same input images as in Section 4.5.4. However, the $P_{16}$ saliency maps, which we display on the right of Figure 4.6.1, illustrate some of the semantic concepts learned by $P$ models. Some of the $P_{16}$ saliency maps resemble those of the ResNet classifier. The left set of images appear to focus primarily on dog legs, with secondary attention placed in general head areas. While the middle images also contain the box-shape area for each dog’s head, they primarily focus (red areas) on background grass or foliage.

We observed a number of peculiar saliency maps that at first seemed to suggest model collapse or overfit. However, when we overlay the saliency maps with 16-piece grids (right-most images in Figure 4.6.1), the purpose of these maps becomes clear. These neurons
Figure 4.6.1: Top: Saliency maps generated from a pretrained ResNet-18 classifier. Bottom: Saliency maps generated from a ResNet-18-based $P_{16}$.

focus on specific edges of the image or borders between two-pieces. Interestingly, we did not observe any high-activating neurons that placed the main saliency foci on intersections of four pieces.

We submit that the distinct differences in these neuron visualizations show that $P$ indeed learns different kinds of features than other models with similar architectures. As a result, $P$-based losses can provide complementary and unique forms of feedback to $G$ during finetuning.

4.7 Conclusion

We introduced PuzzleGAN, an auxiliary puzzle-based loss and end-to-end differentiable models for finetuning pretrained GAN generators. We provided extensive quantitative evaluation of PGGAN and BigGAN finetuning. Our experiments show that PuzzleGAN finetuning is robust to hyperparameter settings and can lead to significant improvements in FID score with relatively few training iterations. We compared samples from the various finetuning methods and observed that PuzzleGAN finetuning can reduce background noise, straighten edges, and
generally improve image fidelity. Additionally, we performed basic network visualization to compare ResNet-18 classifier and $P_{16}$ neuron saliency maps. We observe that $P$ models learn and focus on distinct image features, which allows them to provide novel types of feedback to $G$ during finetuning.
References


[8] Benedict J Brown, Corey Toler-Franklin, Diego Nehab, Michael Burns, David Dobkin, Andreas Vlachopoulos, Christos Doumas, Szymon Rusinkiewicz, and Tim Weyrich. A


Chapter 5

Trained Truncation Trick

Abstract

Recent state-of-the-art generative adversarial networks employ a heuristic noise sampling technique during inference known as the ‘truncation trick’ (TT). TT improves the visual quality of generated output, but measurably reduces image diversity. We introduce a novel pre-generator filter network, the ‘Trained Truncation Trick’ (TTT). TTT trains briefly with pretrained GANs prior to inference to improve visual quality without reducing diversity. We extensively evaluate our approach against various baselines using multiple state-of-the-art GAN architectures and datasets. We demonstrate that our method provides superior FID scores compared to TT while also providing greater output diversity. In addition, we perform a user study with Amazon Mechanical Turk that demonstrates that TTT maintains the original image identity better than TT methods.
5.1 Introduction

Generative Adversarial Networks (GANs) [12] sample noise vectors from high-dimensional distributions to produce diverse and realistic-looking synthetic images. To improve output quality at inference-time, some methods normalize vectors to lie on a unit ball [3, 40] or encourage realistic linear interpolation during training [13]. Recent approaches [5, 20, 21] employ a heuristic called the ‘truncation trick’ (TT), which resamples noise values that fall outside a predefined range. TT adjusts samples to cluster more densely around the distribution mean in order to avoid extreme values that were seen infrequently during GAN training.

As demonstrated in [5, 20, 21, 27], TT generally improves the quality of synthetic outputs. However, TT is known to measurably reduce the diversity of output samples, especially with small truncation ranges [20]. To address these problems, we introduce a simple pre-generator, latent-space filter network, which we call the ‘Trained Truncation Trick’ (TTT). Our method involves a brief post-training, pre-inference finetuning period, and requires only a slight increase in parameters.

Our contributions are as follows:

1. Novel Trained Truncation Trick (TTT) layer for GAN inference.
2. Extensive quantitative comparison of proposed methods with baseline approaches.
3. User study to measure image realism and content preservation across truncation methods.

We now outline the remainder of our paper. In Section 5.2, we review relevant related work. Section 5.3 provides theoretical ground work and introduces our method. Section 5.4 outlines our experimental setup and Section 5.5 reports our various results. In Section 5.6 we conclude and discuss implications of our work.
5.2 Related

**Generative Adversarial Networks** The past decade has produced an enormous amount of improvements and successful applications of GANs. Despite impressive results across a variety of domains, training GANs to produce high-quality and diverse outputs remains a challenging task. In theory, GANs parameterize a two-player game in which the generator and discriminator converge to a minimax solution. In practice, GANs often fail to converge due to exploding and vanishing gradients, mode collapse, or other training hindrances. Explored improvements to GANs include architectural enhancements [20, 29, 35], better model update schedules [16], and loss functions with more stable gradients [2, 13, 29, 38, 39, 46].

5.2.1 Truncation trick precursors

The truncation trick is based on a heuristic sampling approach known as the tempered softmax. The probabilistic recursive super-resolution method of [9] introduced the tempered softmax to adjust a categorical pixel-value distribution, $p$:

$$p_\tau = \frac{p^{1/\tau}}{||p^{1/\tau}||_1}$$  \hspace{1cm} (5.1)

where $\tau \in [0, 1]$ is the temperature parameter. [9] observed improved results with certain temperature ranges, but noted that the the distribution collapses to a single mode as $\tau$ approaches 0. This occurs because of the decreasing entropy as $\tau \to 0$. Later methods [23, 32] also leveraged the tempered softmax and observed similar performance and diversity tradeoffs for different values of $\tau$ (i.e. smaller $\tau$ leads to less diverse output but higher quality individual samples).

5.2.2 Truncation trick

The increasingly common ‘truncation trick’ (TT) is a straightforward extension of the tempered softmax, but modulates values of latent GAN inputs, $z$, to provide a slight performance boost
at inference time. TT aims to avoid areas of low probability [41], such as ‘dead zones’ in large latent spaces that do not lie on the learned data manifold [25].

We observe a number of variations of TT in recent literature. [27, 33] employ a simple truncated uniform distribution (TUD), with \( z \) values samples from \([-\tau, \tau]\). They note the inverse relationship between quality and diversity: Larger \( \tau \) values produce diverse sets of images, while smaller \( \tau \) produce higher quality images at the expense of diversity.

StyleGAN, [20], which builds upon the successful Progressive Growing of GANs [19], takes a two-step approach for TT. They train an 8-layer mapping network to project Gaussian-sampled \( z \) onto latent space \( W \). According to [20], \( W \) provides more linear-like interpolation properties that allow for smoother transitions in generator output. Because this network projection can potentially push \( z_i \) to extreme values, [20] apply TT to \( W_i \) before passing \( W_i \) into the generator. The improved followup to StyleGAN, StyleGAN2 [21], also employs this same TT formulation. Similar to previous work, they note the tradeoff in recall (diversity) and precision (quality) for different \( \tau \) settings of TT.

The authors of the state-of-the-art BigGAN [5] model observe that TT does not extend naturally to some large-scale models. By using orthogonal normalization (ON) [6] during training, however, they are able to employ TT on large models during inference with some success (60% of of their models trained with ON responded well to inference when using TT).

Here we distinguish between the similar descriptions but different implementations of TT in recent work. The uniform subdistribution of [27] is the simplest implementation of TT, and TT can be used with other distributions (e.g. Gaussian). In [20, 21], however, TT consists of a linear interpolation in \( W \)-space based on a given \( \tau \). This mirrors alpha blending in computer graphics or the layer fade-in method of the Progressive Growing of GANs [19]. After mapping \( z_i \) to \( W_i \), StyleGAN’s TT interpolates (rather than truncate in the traditional sense) as follows:

\[
\hat{W} = E[W] + (W_i - E[W]) * \tau
\]  

(5.2)
where $\tau \in [0, 1.0]$ is the interpolation scaling value and $\mathbb{E}[W]$ is the expectation of $W$ computed over 1,000 latent $z$ samples.

As for BigGAN, [5] states that all values outside $[-\tau, \tau]$ are resampled to fall within the desired truncated normal distribution. [36], which uses the official unreleased BigGAN code, clarifies the actual TT behavior: Individual $z$ values outside of two standard deviations of $\tau$ (i.e. $z_i \notin [-2\tau, 2\tau]$) are resampled. In this work, we consider two versions of TT found in recent work. Given a truncation value, $\tau$:

1. TTz: Samples $z$ from a truncated Gaussian distribution with all probability mass concentrated between $[-\tau, \tau]$.

2. TTw: Resamples $W$ values as shown in Eq. 5.2.

Although TT can enhance the aesthetic qualities of outputs, TT noticeably decreases output diversity. For example, the number of possible outputs decreases to 1 as $\tau \to 0$. We now introduce the novel ‘Trained Truncation Trick’ (TTT), which uses gradient descent to learn a fast feed-forward TT replacement.

5.3 Method

5.3.1 Trained Truncation Trick (TTT)

In Figure 5.3.1, we illustrate our novel TTT approach. TTT is a pre-generator latent filtering network that consists of $k$ residual blocks. Drawing inspiration from well-established residual block architectures [15, 45], as well as StyleGAN’s latent mapping network [20], we consider several architectural variations for the blocks within TTT.

The simplest TTT variation mirrors StyleGAN’s 8-layer latent mapping network, where each block consists of a fully connected layer followed by a PReLU [14] activation function (architecture g. in Figure 3). Unlike StyleGAN, we do not aim to learn a completely new latent space with different properties, since this requires extensive training. Instead
we aim to quickly train a latent filter that largely maintains the original latent distribution. We thus connect each of the $k$ layers in our TTT blocks with a residual connection. We also initialize the fully connected weights using normal distribution, $N(0, 1e-9)$, where the density is highly concentrated around 0, and set all biases to 1e-9. Because of this near-zero weight initialization and the residual connections, the untrained TTT mapping network approximates an identity function, i.e., $z \approx TTT(z)$. Empirically, we observe this initial identity-like behavior up to $k = 128$ blocks. TTT with $k > 128$ tends to diverge because, as noted in [47], variance in ResNet output grows exponentially with the number of layers.

Recent work [47] introduces a novel weight initialization method that allows for scaling up to 10,000 residual block layers without traditional batch normalization. Because of our our identity-like TTT initialization, however, we instead turn to common batch normalization-based ResNet blocks [15, 17, 42, 45], which allow for fast and stable learning in deep networks. We first consider three architectures based on the residual blocks in [45] and [17], which we illustrate in Figure 5.3.2 (a-c).
These architectures rely on fully-connected (Linear), batch normalization (BatchNorm), and PReLU activation layers. We comment briefly on two design choices: (1) Following the example of \([20]\), the number of hidden nodes in each fully-connected layer matches that of the input, \(z\). (2) In architectures c. and f., the middle fully-connected layer reduces the dimensionality of the latent space by half. The final fully-connected layer returns \(z\) to its original dimensionality.

While \([17, 45]\) use batch normalization and activation layers prior to the convolutional layer (à la architectures a-c), other successful works \([15, 42]\) use a layer ordering of convolution, batch normalization, and then apply an activation function. We thus evaluate three similar architectures with layers revised in this manner (see d-f in Figure 5.3.2).

**Training** TTT is a network trained prior to inference that functions as a fast single-pass, feed-forward network. To train TTT’s parameters, \(\theta_{TTT}\), we freeze G, D, and any other auxiliary networks involved in training G. Using Adam \([22]\), we update \(\theta_{TTT}\) with \(L_G\), a differential G loss, to learn a TTT filter that encourages high quality samples. Following the example of \([38, 46]\), we use the straightforward Hinge loss \([24, 29, 39]\) formulation of \(L_G\) unless otherwise stated.

Our training requires only a few minutes: We randomly sample latent \(z\) in batches of 10 and train \(\theta_{TTT}\) for 1000 iterations. Our approach is similar to CoachGAN \([7]\), which uses gradient descent at inference time to adjust \(z\). However, \([7]\) requires up to 100 iterations of
repeated gradient descent for a single batch at inference time. TTT is a single pass mapping that pushes latent dimensions toward more favorable regions without the need for random resampling or inference-time optimization.

5.4 Experiments

Models We evaluate 5 official pretrained StyleGAN2 [21] G/D model pairs, which trained respectively on FFHQ [20], and the LSUN cats, cars, churches, and horses datasets [44]. We also use the Progressive Growing of GANs (PGGAN) [19] model and BigGAN [5] models, which trained respectively on CelebA-HQ [19] and ImageNet [37]. We use the official published weights for PGGAN experiments and a publically available PyTorch model for BigGAN experiments\(^1\).

Comparison Methods We compare all models against baseline (i.e. no TT) and TTz. For StyleGAN2 experiments, we also include TTw.

TTT architectures We evaluate TTT with \(k \in [2, 4, 8, 16, 32]\) using the 7 different architectures shown in Figure 5.3.2. We distinguish between TTT that follow immediately after \(z\) and after \(W\) as TTTz and TTTw, respectively.

Training We train each of our models for 1000 iterations with batch of size 10. We use a learning rate of 0.000001 for TTTz and 0.00001 for TTTw.

Metrics We report Fréchet Inception Distance (FID) [16] for all experiments as a measure of output quality and diversity. Following the recommended best practice [4], we use 50,000 samples from each method to compute FID scores.

\(^1\)https://github.com/ajbrock/BigGAN-PyTorch
**Amazon Mechanical Turk** While FID is widely used for image quality assessment in GAN literature, human judgment remains the gold standard in image evaluation. We thus conduct an Amazon Mechanical Turk [31] experiment to measure human opinion of generated outputs from different methods. We discuss full details of this setup and results in Section 5.5.5.

5.5 Results

For all experiments, we report the best TTTz and TTTw results across architectures and $k$ values. In general, we observed that architectures a, c, and g yielded the best results. Interestingly, the d-f architectures, which mirror the widely used DCGAN [35] block architecture, performed poorly in all experiments. We note that TTTz and TTTw tended to yield the best results with $k = 32$ and $k = 2$, respectively.

5.5.1 StyleGAN2 Quantitative Results

In Table 5.5.1, we report the FID results for the StyleGAN2 experiments. Although StyleGAN and StyleGAN2 do not evaluate TTz, we included this approach in our experiments. We highlight several patterns of interest in the FID scores across the five datasets and methods: TTw noticeably increases (worsens) FID scores, in some cases more than doubling the score (i.e. LSUN church, cat, and horse). TTz generally increases FID with the exception of the LSUN Car dataset. In contrast, TTTz improves FID across the board, yielding 0.1-0.43 improvements.

TTw, the official truncation trick for StyleGAN2, significantly worsens FID because TTw trades quality for diversity, as noted by StyleGAN authors [21]. TTTw increases FID, but to a much smaller extent than TTw. For instance, TTw yields an FID score of 16.43 for LSUN Church, while TTTw produces a score of 8.84. This pattern repeats across all
datasets. To better understand this trend, we explore differences between TTw and TTTw qualitative samples in Section 5.5.2.

5.5.2 StyleGAN2 Qualitative results

**LSUN car** In Figure 5.5.1, we compare sampled images from the various methods generated using the same initial $z$ vectors. In the left-most column, we display the outputs generated without any truncation. As demonstrated by the first row, TTz and TTw can significantly alter the content of the image. Across our StyleGAN experiments, we observe that TTTz preserves original content features with the greatest fidelity. This helps maintain image diversity which, when coupled with improved realism, leads to improved FID scores.

For instance, the first row shows that TTTz clarifies the license plate and some of the car details, but otherwise retains identifiable features of the car. TTz and TTz completely alter the make and direction of the car. TTTw affects outputs more than TTTz, particularly the coloring of the image. This occurs because TTTw adjusts latent $W$, which thereafter feeds into and modulates the outputs of various network layers of the G synthesizing network. This leads to more noticeable stylistic changes in outputs. However, TTTw preserves image diversity and content better than TTw, as reflected in the FID scores in Table 5.5.1.

**FFHQ** In Figure 5.5.2, we display image samples from the various methods in the same format as Figure 5.5.1. We observe that TTz and TTTz tend to preserve the original face features more than TTw and TTTw. As the second and third rows demonstrate, TTTz
tends to better handle partial occurrences of glasses. For instance, the single glass lens over the man’s right eye is removed by TTTz, while TTz retains it. TTz removes the partial frame over the woman’s left eye, but also changes the woman’s hairstyle and eye size. TTTz, however, completes the glasses while better maintaining the woman’s identifying characteristics.

**LSUN cat** The LSUN cat samples in Figure 5.5.3 show clearly the feature preserving advantages of TTTz and TTTh. In addition, TTTz provides more realistic enhancements to the original output than TTz (e.g. completing the cat’s head in the bottom row, clarifying the cat’s limbs in the second row, etc.). This demonstrates the advantage of a learned truncation compared to random resampling or using a truncated distribution.
Figure 5.5.2: StyleGAN2 FFHQ samples using no truncation (original), TTz, TTTz, TTw, and TTTw.

Figure 5.5.3: StyleGAN2 cat samples using no truncation (original), TTz, TTTz, TTw, and TTTw.
Table 5.5.2: Progressive Growing of GAN FID results. We evaluate them against the FID statistics computed over FFHQ, since CelebA-HQ only contains 30k images.

<table>
<thead>
<tr>
<th>Method</th>
<th>FID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>66.98</td>
</tr>
<tr>
<td>TTz</td>
<td>64.79</td>
</tr>
<tr>
<td>TTTz</td>
<td>63.99</td>
</tr>
</tbody>
</table>

5.5.3 PGGAN

In addition to StyleGAN2, we evaluated our truncation methods using the Progressive Growing of GANs [19]. In Table 5.5.2, we report FID scores for baseline, TTz, and TTTz. TTTz using architecture g and $k = 4$ (see Figure 5.3.1) produced the best FID score. Specifically, TTTz improves (decreases) FID of baseline and TTz by 2.99 and 0.8, respectively.

In Figure 5.5.4, we compare the effects of using more layers (top) and longer iterations (bottom) in TTTz with architecture g. We show the effects of training $k \in [2, 4, 8, 16, 32]$, as well as a 2 layer TTTz for 1k-10k iterations. As iterations increase, TTTz pushes different starting $z_i$ toward a single mode. As the number of layers increase, we see changes that start to reflect biases of the data (face forward, no glasses), but better maintain image identity. We observed these patterns across hundreds of other qualitative samples.

5.5.4 BigGAN

We compared TTz and TTTz using BigGAN [5] trained for 138k iterations on ImageNet. We experimented with $\tau \in [0.1, 0.2, \ldots, 1.0]$ for TTz and report the best results, which we obtained with $\tau = 0.4$. We present the FID results for this experiment in Table 5.5.3.

Unlike StyleGAN2 and PGGAN, we do not observe an improvement in FID scores. This is not entirely unexpected, since BigGAN is known to not always respond well to the truncation trick [5]. Additional architecture modification to TTT blocks (e.g. adding
Figure 5.5.4: Comparison of the effects of the number of TTTz layers and the number of training iterations.
Table 5.5.3: BigGAN FID results

<table>
<thead>
<tr>
<th>Method</th>
<th>FID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>38.64</td>
</tr>
<tr>
<td>TTz ($\tau = 0.4$)</td>
<td>37.28</td>
</tr>
<tr>
<td>TTTz</td>
<td>38.61</td>
</tr>
</tbody>
</table>

orthogonal normalization as in [5]) may enable the successful use of TTTz. However, we leave this exploration for future work.

5.5.5 User study

An increasing number of GAN publications employ Amazon Mechanical Turk (AMT) [31] to assess human preference for different generated outputs [1, 8, 26, 43]. We similarly conduct an experiment on AMT using a StyleGAN2 [21] generator trained with FFHQ. We sample images using identical random seeds which allows us to compare the effects of different truncation methods.

While online survey tools such as AMT provide an unprecedented level of access to human opinion, they present their own unique challenges. Since AMT respondents receive payment for completed surveys, there is an inherent bias toward ‘satisficing’ [10, 30], i.e., responders put minimum effort into the survey to meet the basic completion requirements. This is a well-documented occurrence that skews and otherwise negatively impacts the quality of collected data [18, 30, 34].

To guard against poor quality AMT responses, we take the following precautions: We require AMT workers to have an approval rating greater than 97% and have over 5,000 previously completed survey tasks. This limits our survey to workers who have an established history of doing high quality work. We also use a simple but inconspicuous trap question in each survey to validate human effort [11, 28]. We eliminate all responses that do not correctly answer the trap question in order to reduce tainted data from satisficing workers.
Setup We sampled 1,000 StyleGAN2 FFHQ images using identical seeds with no truncation (i.e. original), TTz, TTTz, TTw, and TTTw. For each image seed set, we randomly select and order two images on the user interface. We ask the worker to drag a slider toward the most realistic-looking image. The slider allows 0-100 values where 0 indicates that the left image looks most realistic, 50 indicates equal levels of realism, and 100 indicates that the right image looks most realistic.

Next, we present a user with an original (no truncation) image and randomly sample and order two images from TTz, TTTz, TTw, and TTTw methods. Users move another 0-100 slider toward the image that best preserves the original image content. Figure 5.5.5 illustrates this slider setup and displays the results for the content preservation user study.

In Tables 5.5.4(a) and 5.5.4(b), we report the mean slider values with 95% confidence intervals for the realism and content experiments, respectively. Table 5.5.4(a) shows that TTw improves image realism more than other truncation methods. However, this improvement comes at the expense of diversity and content preservation, as shown in Table 5.5.4(b). TTz and TTTz show relatively similar levels of realism, with a slight preference toward TTz. The content results in Table 5.5.4(b) provide a complementary view of the various methods. Compared to traditional truncation methods, TTTz and TTTw preserve image content significantly better. This reinforces the takeaways from the qualitative results presented in
Table 5.5.4: Mean slider values with 95% confidence intervals for (left) realism and (right) content preservation user studies on Amazon Mechanical Turk

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Slider Value</th>
<th>Comparison</th>
<th>Slider Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>orig-TTz</td>
<td>49.77 ± 1.77</td>
<td>TTz-TTw</td>
<td>28.04 ± 1.76</td>
</tr>
<tr>
<td>orig-TTw</td>
<td>65.68 ± 1.66</td>
<td>TTz-TTTw</td>
<td>68.99 ± 1.95</td>
</tr>
<tr>
<td>orig-TTTw</td>
<td>47.87 ± 1.55</td>
<td>TTz-TTTz</td>
<td>76.38 ± 1.75</td>
</tr>
<tr>
<td>TTz-TTTz</td>
<td>50.90 ± 0.98</td>
<td>TTw-TTTw</td>
<td>82.50 ± 1.70</td>
</tr>
<tr>
<td>TTz-TTTw</td>
<td>67.69 ± 1.55</td>
<td>TTw-TTTz</td>
<td>84.02 ± 1.31</td>
</tr>
<tr>
<td>TTw-TTTw</td>
<td>42.85 ± 1.84</td>
<td>TTTw-TTTz</td>
<td>73.07 ± 1.71</td>
</tr>
<tr>
<td>TTw-TTTz</td>
<td>45.68 ± 1.80</td>
<td>TTTw-TTTz</td>
<td>73.07 ± 1.71</td>
</tr>
<tr>
<td>TTTw-TTTz</td>
<td>32.63 ± 1.77</td>
<td>TTTw-TTTz</td>
<td>73.07 ± 1.71</td>
</tr>
<tr>
<td>TTTw-TTTz</td>
<td>30.78 ± 1.70</td>
<td>TTTw-TTTz</td>
<td>73.07 ± 1.71</td>
</tr>
<tr>
<td>TTTw-TTTz</td>
<td>48.51 ± 1.62</td>
<td>TTTw-TTTz</td>
<td>73.07 ± 1.71</td>
</tr>
</tbody>
</table>

Section 5.5.2. Namely, TTz and TTw can provide improvements in realism at the cost of identity and diversity. TTTz and TTTw largely preserve identity while still improving the aesthetic qualities of images.

5.6 Conclusion

In this work, we introduced a novel latent-space filter network, the ‘Trained Truncation Trick’ (TTT). We demonstrated the superiority of TTT over existing truncation approaches through extensive FID evaluations with multiple GAN architectures and datasets. We further validated our approach with AMT user studies. We found that compared to traditional truncation, TTT preserves content and image diversity to a much greater extent. Future work will explore additional architectures, specifically those that enable successful use of TTT with BigGAN. Future work will also look into dynamically enabling network capabilities at inference time via TTT.
References


[10] Carol Galais and Eva Anduiza. “you cheated on me!” causes and consequences of cheating in online surveys. Visions in Methodology workshop.


Chapter 6

Two Second StyleGAN Transfer

Abstract

We introduce a Fast StyleGAN2 Projection (FSP) algorithm, which efficiently embeds arbitrary real images into the latent-noise space of StyleGAN2. Compared to existing approaches, we reduce the time needed for high-quality projections by two orders of magnitude, reducing the time needed from 20 minutes to under 2 seconds. FSP achieves this speedup by using a learned latent initialization, simplified loss setup, and large learning rates. We quantitatively compare FSP against state-of-the-art StyleGAN2 projection using LPIPS, PSNR, and SSIM. Our results demonstrate FSP’s superior performance across a wide variety of datasets. We demonstrate the effectiveness of our projection method and highlight new applications enabled by our approach.
6.1 Introduction

Convolution-based Generative Adversarial Networks (GANs) can synthesize novel and diverse images [15, 17, 28, 43, 52, 53], videos [3, 6, 38, 41, 44, 50], 3D objects [23, 24, 36], text [8, 37], and other kinds of outputs [7, 9]. In recent years, the StyleGAN family of architectures [18, 19] have attracted widespread attention due to their realistic, high-resolution outputs with smooth interpolation qualities. An active branch of StyleGAN2-based research learns interpretable latent directions based on domain-restricted, generated data [13, 32, 33, 40]. Despite impressive results, these methods do not address the more challenging task of performing image edits on arbitrary pre-existing images.

While the majority of GAN literature focuses on latent-to-output synthesis, an increasingly popular approach inverts this direction by embedding real images into the flexible StyleGAN2 [19] latent space. This image-to-latent inversion offers the possibility of performing common GAN-based image manipulations on real image inputs.

StyleGAN2 projection has gained traction with the research community and provides increasingly realistic reconstruction and editing capabilities for real images. However, recent approaches [1, 2] can require more than 20 minutes using a state-of-the-art GPU to project a single 256x256 image. This limits the scalability of projection methods – both in terms of image throughput and latency for real-time user interactivity. These drawbacks have spurred various encoder-decoder based alternatives [27, 40] that consume the provided image and predict the corresponding latent code in a feed-forward fashion. While adversarially trained encoder networks such as [27] generate high-quality images, they do so with a significant loss of fidelity to the original inputs, which severely limits the usefulness of such methods for editing user-provided images. This tradeoff mirrors common generator issues of image quality versus diversity (i.e. precision versus recall) [20].

We introduce Fast StyleGAN2 Projection (FSP), which enables efficient, high quality image projection. Our method uses a smaller latent space than state-of-the-art projection
methods, and reduces the time for high resolution projections from more than 20 minutes to under two seconds! This multiple order-of-magnitude speedup stems primarily from our use of a ‘learned latent initialization’, which involves using first image’s projection as starting point for subsequent projections. We also employ a simplified loss setup, large learning rate, and a border-cropping technique to further reduce convergence time.

In our experiments, we demonstrate smooth interpolation and image editing properties on sets of FSP-projected real images. Unlike existing projection methods, which confine interpolation and image editing capabilities to the domain of the pretrained StyleGAN2 generator (e.g., FFHQ faces), FSP can project and edit arbitrary images. Figure 6.1.1 displays examples of images projected in two seconds using our approach.

6.2 Related

6.2.1 Generative Adversarial Networks

Notable improvements in GAN training [17, 31], architecture [5, 18, 46], losses [12], and image projection [1, 2, 40] enable increasingly realistic generation as well as editing of existing images [33]. Various approaches [18, 51] identify disentangled GAN latent spaces after training
unconditional generators. These disentangled spaces enable output interpolation, mixing, and editing functions [11, 26].

More recent approaches impose [35] or learn [21] a latent code space to separate factors of variation in an unsupervised manner. These enable realistic generation conditioned on several attributes using a multi-stage, StackGAN-based [47, 48] approach. Conditional GANs with stronger levels of supervision in the form of semantic labels [42], image-to-image pairs [14], or dense annotations [34], can provide more fine-grained control over generated outputs. However, these methods are confined to synthetically generated outputs and cannot perform semantic or textural edits on real images.

6.2.2 StyleGAN2

Using a trained feed-forward network, StyleGAN2 maps a randomly-sampled, normally-distributed latent $z$ to an intermediate latent code $w$, where $\mathbb{R}^z = \mathbb{R}^w$. The latent $w$ feeds into various generator layers via learned affine transformations, which provide tailored stylizations of intermediate latent activations. StyleGAN2’s generator network also broadcasts a single channel $4 \times 4$ noise map over the first StyleGAN2 block, and two $n \times n$ noise maps for subsequent resolution blocks $\{8 \times 8, 16 \times 16, \ldots, 1024 \times 1024\}$. Formally, the generator broadcasts a noise sample $N_i \in \mathbb{R}^{B \times 1 \times H \times W}$ over $C$ channels, scales each noise map by a learned channel weight, $\gamma_c$, and adds the resulting noise tensor to a generator latent output $L_i \in \mathbb{R}^{B \times C \times H \times W}$. The injected noise improves the separability of the generator’s latent factors and enables realistic interpolation over output content and style.

6.2.3 Image Projection

Existing projection methods [1, 2, 40, 45] leverage the diverse array of realistic images that lie on the learned manifold of pretrained StyleGAN2 generators. Given a real image as input, projection uses convex optimization to search for latent values that allow the generator to reproduce the original real input image. This differs from [39], which optimizes randomly
initialized network parameters. StyleGAN2 projection methods optimize only latent and/or noise inputs – leaving pretrained network weights untouched.

We briefly summarize recent projection approaches. The projection method outlined [1] project images onto an extended $W^+$ latent space. This approach optimizes a unique $w$ vector for each convolution layer inside the various generator style blocks. While effective, the $W^+$ space does not encode fine-grained image details.

The improved projection approaches in [1, 2] leverage extended StyleGAN2 latent-noise spaces, $WN$ and $W^+N$, to allow realistic editing and image styling via latent code mixing. The noise space, $N$, refers to the random noise maps used in StyleGAN2. To illustrate the advantages of the noise space, we project a real face using the original StyleGAN2 latent space, $W$, the extended latent space, $W^+$, the latent noise space, $WN$, and the extended latent noise space, $W^+N$ (see Figure 6.2.1). The $W^+N$ latent space enables high-fidelity reconstruction of projected images. This offers an advantage over adversarial encoder-decoder approaches such as [27], which preserves high level attributes (e.g. hair length, background color, makeup style), but omits basic identifying characteristics (e.g. bone structure, eye color).

Despite the precision of $W^+N$ projections, we emphasize a number of drawbacks that limit extensions for large-scale GAN training: First, a faithful reconstruction of a single image can require more than 20 minutes on an NVIDIA Tesla V100 GPU. Projection also involves a random initialization of the latent-noise space for each image. This results in a one-to-many mapping from each image to possible projections, which hinders interpolation and image editing capabilities. A concurrent work, [45], reduces the spread of projected latent

Figure 6.2.1: StyleGAN2 image projections. From left to right: $W$ projection, $W^+$ projection, $WN$ projection, $W^+N$ projection, original image.
distributions by constraining projections to lie near the mean latent prior used for training the generator (e.g., a zero-centered normal Gaussian for FFHQ faces). Similar to traditional projection approaches, however, this still requires extended optimization periods and limits image editing applications to the data domain of the pretrained StyleGAN2 generator.

6.3 Method

6.3.1 Fast StyleGAN2 Projection

Projection methods perform best when the input image closely matches the style and content of original GAN training domain (e.g., centered face images for a StyleGAN2 generator pretrained on the FFHQ dataset). To the best of our knowledge, no existing work has attempted to project and edit sets of arbitrary images that do not closely correspond to classes learned by the generator.

In order to rapidly project arbitrary sets of real images, we introduce the Fast StyleGAN2 Projection (FSP) algorithm. FSP modifies the projection method of [2] to allow $W^+N$-quality projections in under two seconds. We achieve this order of magnitude speedup by using the smaller WN space with a learned latent initialization, simplified loss, large learning rates, and other optimization improvements. We use the official pretrained StyleGAN2 FFHQ generator weights\(^1\) for all projection experiments, \textit{regardless of the input domain}.

In order to learn a good latent initialization for fast projection of novel images, we randomly sample and project a single input image for 7,500-15,000 optimization cycles with a large learning rate, such as $\eta = 3.8$ (significantly larger than [19]'s original projection method with $\eta = 0.1$). This large $\eta$ produces a starting WN projection, WN\(_{\text{INIT}}\), with near-imperceptible differences in reconstructed output from a high-quality $W^+N$ projection. Instead of randomly initializing the WN latent-noise space for subsequent projections, we use WN\(_{\text{INIT}}\), the initial extended iteration projection, as a starting point for all subsequent

\(^1\)https://github.com/NVlabs/stylegan2
projections. This ‘learned initialization’ allows future projections to converge to high quality
WN embeddings in just 50 iterations (i.e., \( \sim 2 \) seconds on a single NVIDIA Tesla V100 GPU).
We quantitatively compare the image reconstruction quality and runtime of \( \text{W}^+\text{N} \) and FSP in
Section 6.4.4. Although WN is strictly more limited than \( \text{W}^+\text{N} \) in terms of representational
capacity, FSP generally outperforms \( \text{W}^+\text{N} \) projection across a variety of metrics.

FSP uses a straightforward MSE loss in the output space and excludes common style
[10], perceptual [16], and LPIPS [29] losses used in existing projection approaches [1, 2, 29].
Although pixel-wise MSE is notorious for encouraging blurry reconstructions in generative
models [25], we observe empirically that trained StyleGAN2 networks serve as a powerful
natural image prior for projection onto the latent space. Our setup thus echoes the work of
[39], which optimizes the loss

\[
L = \min_\theta ||f_\theta(z) - x_0||^2
\]  

(6.1)

where \( x_0 \) is a corrupted image, \( \theta \) is the randomly initialized parameters of a neural network,
\( f \), which has a architecture that serves as a natural image prior. Unlike [39], however, we
hold network parameters constant and only optimize StyleGAN2’s noise maps and latent
input.

Algorithm 1 describes FSP for a set of arbitrary input images, \( I \). FSP outputs a set
or projections, where \( \text{WN}_i \) corresponds to input \( I_i \). We reiterate that only the first image
projection will optimize for 7,500+ iterations. Other images use the result from the first
projection, \( \text{WN}_{\text{INIT}} \), as a starting point. This allows all subsequent projections to converge in
just 50 iterations.

A concurrent work, [45], encourages \( \text{W} \) and \( \text{W}^+ \) projections to lie near the latent prior
of the pretrained generator. This principle extends naturally to WN and \( \text{W}^+\text{N} \). However, we
found empirically that adding a latent prior term in the optimization loss limits convergence
and produces poor \( \text{WN}_{\text{INIT}} \) and \( \text{WN}_i \). Because of this, we do not constrain the latent
Algorithm 1: Fast StyleGAN2 Projection (FSP)

**Input:** \( I = \{I_1, I_2, \ldots, I_n\} \)

**Output:** \( \{WN_1, WN_2, \ldots, WN_N\} \)

\[ WN_{\text{INIT}} \leftarrow \text{Project}(I_1) \text{ for } 7,500+ \text{ iterations;} \]

\[ \text{for } i \leftarrow 2 \text{ to } n \text{ do} \]

\[ WN_i \leftarrow WN_{\text{INIT}}; \]

\[ WN_i \leftarrow \text{Project}(I_i) \text{ for } 50 \text{ iterations;} \]

\[ \text{end} \]

<table>
<thead>
<tr>
<th>( W_{i_1} )</th>
<th>( W_{i_2} )</th>
<th>( W_{i_3} )</th>
<th>( W_{i_4} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu )</td>
<td>31.8</td>
<td>-53.2</td>
<td>34.1</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>1.93</td>
<td>0.62</td>
<td>1.24</td>
</tr>
</tbody>
</table>

Table 6.3.1: Mean (\( \mu \)) and standard deviation (\( \sigma \)) of the first four consecutive dimensions of \( W \) for fourteen scene images projected using FSP (the bottom row of Figure 6.3.1 shows example images). The means vary widely, while standard deviations show comparatively small differences.

optimization of \( WN_{\text{INIT}} \) beyond our described MSE loss setup. This approach allows FSP to project various images – potentially from many different domains – into a similar latent space. Instead of embedding projections around the generator latent prior, \( WN_{\text{INIT}} \) serves as the learned prior for all subsequent \( WN_i \) projections.

### 6.3.2 Properties of FSP latent space

Individual latent dimensions of FSP-projected \( WN_i \) exhibit low standard standard deviations across all projections. However, the means of different dimensions within the same noise layer or \( W \) (e.g., \( w[j] \) and \( w[j+1] \)) can vary widely. Table 6.3.1 compares the differences in the mean, \( \mu \), and standard deviation, \( \sigma \), of the first four dimensions of \( W \) for a set of 14 unrelated scene images projected with FSP. These statistics suggest that FSP finds a flexible projection space, \( WN_{\text{INIT}} \), which adapts easily to represent a wide variety of unique images.

To demonstrate the flexibility of our projection method, we project a video from the FaceForensics [30] dataset where the subject’s head size and eye location do not match the eye-aligned FFHQ training instances. We also project unrelated scene images (bottom two...
Figure 6.3.1: The 1st and 3rd rows show original images, and the 2nd and 4th rows show WN projections using our FSP method. The far left images in the projected outputs are optimized for 1500 iterations to generate the WN\textsubscript{INIT}. The remaining projections are trained for 100 iterations (under two seconds) using WN\textsubscript{INIT} as the latent initialization.

rows of Figure 6.3.1) to demonstrate the robustness of our projection method with diverse types of inputs. All projections use the official StyleGAN2 FFHQ pretrained generator weights.

We emphasize that the quality of WN\textsubscript{INIT} strongly impacts ensuing projections. Figure 6.3.2 compares the effects of low and high-quality WN\textsubscript{INIT}: The top row displays the image reconstructions for WN\textsubscript{INIT}, and the bottom row contains reconstructions from fast projections. The noisy initial projection (left column) produces a fast projection with obvious noise artifacts in the reconstructed bottom image. The high-quality initial projection, on the other hand, yields fast projections with considerably less residual noise.

6.3.3 Inverted Layer Importance

Unlike StyleGAN2 generation - where lower layers affect broad content changes and higher layers influence lighting and color - early FSP noise maps affect small and often negligible changes in output coloring. Middle noise maps gradually add background details, and mid-
Figure 6.3.2: Top: Image reconstructions for WN\textsubscript{INIT} (the extended, ‘learned latent initialization’). Bottom: Image reconstructions for fast projections. Each bottom image uses the corresponding top image’s WN\textsubscript{INIT} as a starting point. The quality of WN\textsubscript{INIT} significantly impacts the quality of subsequent fast projections.

Figure 6.3.3: Left: Original images. Right: The 7th FSP-projected noise map, N\textsubscript{7}, encodes edge details of the foreground object(s). By swapping the 7th FSP-projected noise maps, N\textsubscript{A7} and N\textsubscript{B7}, from two different FSP projections A and B, we can generate a new image with the edge details of B’s foreground and the background of A.

to-upper noise maps fill in foreground edge details, coloring, and textures. In Figure 6.3.3, we swap the 7th noise map, which controls foreground edge details, of two different FSP projections that use the same WN\textsubscript{INIT}. This produces an edge-outline image of a bird from one scene with the background of the other scene.

We further analyze the various noise maps by projecting the Magpie image subset from ImageNet using FSP and performing PCA independently on W and each noise map. While controllable interpolation on the scale of [13, 32] goes beyond the scope of this work, we identify components in the 7th, 8th, and 10th noise maps that control the brightness, sharpness, and opacity of projected images (see Figure 6.3.4). Due to the decoupling of
Figure 6.3.4: We perform Principal Component Analysis on the 7th, 8th, and 10th noise maps of sizes 32x32, 64x64, and 128x128, respectively. Movement along the first components of these PCA decompositions (respectively) changes aspects like opacity, brightness, and sharpness in the output image.

foreground edge and color details, we cannot interpolate image content using a single noise map. However, we explore content interpolation using PCA computed over flattened and concatenated WN in Section 6.4.4. We leave non-linear WN decomposition and conditional mapping as a promising direction for future work.

6.3.4 Image Normalization

In addition to a high-quality WN\textsubscript{INIT}, image normalization statistics (i.e. those used to map pixels from the range [0,255] to [-1,1] prior to projection) can impact color fidelity, image detail, and noise artifacts in reconstructed images. The impact of normalization on color is particularly noticeable when projecting consecutive video frames that contain the same background: Single image normalization captures finer details at the cost of a ‘flickering background color’ effect among adjacent video frames. By normalizing all frames with the same statistics, however, backgrounds of images will maintain consistent coloring across frames. We use the latter normalization approach for the FSP example in the second row of Figure 6.3.1.
While helpful for maintaining consistent image coloring, batched or dataset-wide normalization can increase noise artifacts when the color distributions vary widely or contain ‘unnatural’ gradient flows. For instance, black image borders (natural or artifacts from scanning) skew normalization statistics and lead to visible noise in reconstructed images. FSP can remove the undesired noise with additional optimization or appropriate image preprocessing, which we describe in the following section.

6.3.5 Training

In addition to our learned initialization, large learning rates, and MSE loss setup, we describe additional training settings for experimental reproducibility. These settings yield a smaller yet noticeable impact on projection quality.

We remove the common learning rate ramp-up (used in [19]'s original projection approach) and reduce the linearly annealing learning rate ramp-down period from the last 25% of iterations to 3%. Although common practice, we observe that ramp-ups and longer ramp-down periods slow FSP’s convergence without improving reconstruction quality.

For the initial extended projection, we add random noise $\zeta \sim \mathcal{N}(0, 0.001)$ to the target image at each optimization iteration. Without this regularization, MSE converges more easily to local minima, which produces spatters of unnatural, colored noise throughout the image reconstruction. We remove the $\zeta$ regularization for all subsequent projections.

Unlike WN and W³N StyleGAN2 projection [2], we also remove the regularization loss term on the noise maps. We experimented with an L1 regularization to push fast projections toward the initial learned projection. However, this prior regularization nearly doubles the number of iterations required to produce high quality visual reconstructions. Instead, we use a large learning rate, $\eta = 3.8$, for WN$_{\text{INIT}}$ and $2\eta$ for WN$_i$, which only optimize for 50 iterations. This naturally embeds all projections near WN$_{\text{INIT}}$ while still producing high quality output reconstructions.
We remove the outer 3 rows and columns of border pixels in reconstructed images prior to computing MSE. This alleviates the need for FSP to model sharp gradient changes that often occur along input image borders. The border removal also simplifies projection for convolution-based GANS (i.e. StyleGAN2), which can struggle to generate accurate borders due to zero-padding of image inputs.

In Figure 6.3.5, we illustrate the discoloration and noise that standard projection \((W_+ N)\) introduces along border regions. By cropping the outer borders prior to computing MSE, we reduce convergence time and improve the overall color agreement between the projected and original images.

### 6.4 Experiments

#### 6.4.1 Image Interpolation

Interpolation between one or more out-of-domain latent \((W,W_+)\) or latent-noise \((WN,W_+N)\) projections produces ‘phantom’ artifacts from the pretrained generator domain. This occurs because interpolations between independent projections may cross over the generator latent prior, which is biased toward generating certain kinds of outputs. For example, a StyleGAN2 generator trained on FFHQ will naturally yield face-like artifacts [1].

We illustrate this problem in Figure 6.4.1, which compares \(W+N\) interpolations between latent projections using the the method outlined in Image2StyleGAN++ (I2S++) [2]. We provide interpolations for the LSUN bedroom and CelebA datasets. The I2S++ interpolations
regress toward outputs of the original FFHQ domain, as evidenced by the unnatural faces that appear in the middle images of each row.

Unlike $W^+N$, FSP interpolates in an isolated, latent projection space centered on $WN_{\text{INIT}}$. This allows FSP to interpolate between projections without out-of-place face artifacts. Figure 6.4.2 shows sample interpolations for the LSUN and CelebA datsets.
Figure 6.4.2: Interpolations between FSP projections of LSUN bedroom (top) and CelebA (bottom) samples. FSP avoids the ‘phantom faces’ that appear in $W^+N$ interpolations in Figure 6.4.1.
6.4.2 PCA Interpolation

By mapping small changes in the latent space to semantically meaningful updates in the output space, GANs can provide realistic interpolations using simple linear interpolation methods [4]. Recent work shows that PCA decompositions over latent activations can further improve interpolation quality. For instance, [13] perform PCA over various StyleGAN2 layer output activations to introduce interpretable controls for synthesizing new output.

Drawing inspiration from this line of work, we perform PCA decomposition on FSP projections for the CIFAR-10 test set, which consists of 10,000 32x32 images from 10 different classes. We also perform PCA interpolation on the Magpie subset of images from ImageNet, which we rescale and crop to 128x128. Figures 6.4.3 and 6.4.4 show sample PCA interpolations between CIFAR-10 and Magpie FSP projections, respectively. We use the $k$ components that explain up to 95% of the total variance. We find empirically that this approach improves the quality of interpolation and reduces the noise-level in intermediate interpolation outputs.
6.4.3 Style Transfer

In FSP projections, $W$ most strongly influences color and high level texture details in image reconstructions. This allows for basic style transfer between two different FSP projections that optimize using the same $W_{\text{INIT}}$. In Figure 6.4.5, we perform PCA interpolation between two unique FSP projections of Magpie images. Unlike Section 6.4.2, however, we do not interpolate $W$. Instead, we use the $W$ of the far right image’s FSP projection for all image reconstructions. This transfers textures and color details from the rightmost image to the other image reconstructions.
Table 6.4.1: Quantitative comparison of W+N projection and FSP. For LPIPS we report both AlexNet (left) and VGG (right) scores in the same column.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>W+N</th>
<th>FSP (Ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSNR↑</td>
<td>SSIM↑</td>
</tr>
<tr>
<td>Scenes</td>
<td>28.367</td>
<td>0.8785</td>
</tr>
<tr>
<td>ImageNet-M</td>
<td>24.1506</td>
<td>0.7171</td>
</tr>
<tr>
<td>CelebA-S</td>
<td>29.705</td>
<td>0.9026</td>
</tr>
<tr>
<td>LSUN-S</td>
<td>22.143</td>
<td>0.6439</td>
</tr>
<tr>
<td>Cityscapes-S</td>
<td>25.880</td>
<td>0.8616</td>
</tr>
</tbody>
</table>

6.4.4 Quantitative Reconstruction Quality

We use PSNR, SSIM, and LPIPS [49], common methods for reference-based image quality assessment [22], to quantitatively compare projected image reconstructions from the current state-of-the-art method, W+N [2], and FSP. For LPIPS, we include results for both AlexNet and VGG as the base feature extraction network. We exclude feed-forward methods such as [40] and [32], since they focus on closed-domain semantic editing (i.e. face feature adjustments), rather than generalized image projection. For these experiments we use the LSUN bedroom, CelebA, Cityscapes, and ImageNet datasets.

Due to the extended runtime of W+N projections, we do not project the full datasets. Instead, we allow W+N to project images within a given dataset for up to 48 hours, using 5,000 optimization iterations per image on an NVIDIA V100 GPU. We then use FSP to project the same subsets of ground truth images projected by W+N in the 48-hour periods. This allows us to compare 99 LSUN (256x256), 99 Cityscapes (256x256), 144 CelebA faces (128x128), and 138 Magpie images (128x128) from ImageNet. We also compare 15 unrelated scene images (512x512) that contain a variety of different light settings, textures, and content.

For the Cityscapes experiments, we project finely annotated semantic and panoptic label maps from the Frankfurt data subset. This provides a more challenging task for StyleGAN2-based projection, which more easily handles natural images with generally smoother gradients, rather than sharp pixel-wise annotations in the semantic label maps.
We present results for these experiments in Table 6.4.1. In the LPIPS columns, we report results for AlexNet-based LPIPS followed by VGG-based LPIPS. Across the twenty metrics reported for each method in Table 6.4.1, FSP outperforms W+Na in all but six cases. Four of these cases involve VGG-LPIPS. Since W+Na projection optimization involves a VGG-based perceptual loss term, we expect better performance of W+Na when using VGG-based LPIPS. For AlexNet LPIPS, however, FSP outperforms W+Na across all datasets excluding ImageNet.

In Figure 6.4.6, we compare the samples of original images with projected W+Na and FSP outputs for the CelebA, Cityscapes, and LSUN datasets. Given 5,000 optimization iterations (∼20 minutes) for each image, W+Na fails to encode fine details of images. By contrast, FSP captures color and low-level content details – albeit with traces of residual pixel noise – in just 2-4 seconds.

6.5 Conclusion

We introduce Fast StyleGAN Projection (FSP), a method that provides an order-of-magnitude time reduction for StyleGAN2 image projection compared to state-of-the-art W+Na projection. We demonstrate FSP on a variety of datasets with various resolutions, as well as content and style diversities. FSP consistently yields high quality results, as we demonstrate with numerous qualitative samples as well as LPIPS, PSNR, and SSIM comparisons.

We anticipate fast-er StyleGAN projection methods in future work. In Section 6.3.3 we identify the inverted importance of layers in projections. This suggests that StyleGAN2 projection can operate in a reduced or pruned space, where some of the noise maps are ignored entirely during optimization and image reconstruction. In fact, we observed that only the 512-dimensional W vector, the second 32x32 noise map, and the first 128x128 noise map (a total of 17,920 parameters) are essential for 128x128 Magpie projection reconstructions. For the remaining 26,986 out of 44,176 WN parameters, we found that replacing noncritical
Figure 6.4.6: Projection comparisons of Cityscapes, CelebA, and LSUN Bedroom. For each dataset, we display the original (top), W$^+$N (middle), and FSP (bottom) projections. FSP reconstructs images more accurately than W$^+$N, which struggles to encode fine details and sharp gradient changes.
noise maps with random Gaussian noise did not visually affect FSP image reconstruction quality. This suggests paths for additional optimization improvement.

We expect future explorations to enable fast discovery of disentangled FSP latent representations. This could allow for more fine-grained image-to-image translations (i.e. semantic to panoptic, original to semantic, etc.) via projection, latent editing, and image reconstruction. By significantly reducing projection time, FSP enables large-scale projection of datasets of related or unrelated sets of images. This will allow for more advanced latent-space editing techniques on real, projected images.
References


Chapter 7

Conclusion

In this work, we introduced several algorithms for finetuning and improving the quality of generated GAN output. We presented a wide array of qualitative results and quantitative evaluations across models and datasets to support our claims. Our early work in Chapter 2 addressed the issues of stability and diversity in Multiple Choice Learning (MCL). We identified and introduced solutions to mitigate Alpha Model Domination in MCL ensemble training.

Thereafter we focused on inference time improvements for GANs, which naturally support the diverse output aims of MCL. We leveraged the densely packed information in latent inputs and early generator layers to yield measurable and visually identifiable output enhancements for GANs at inference time. Specifically, we introduced CoachGAN in Chapter 3 as a latent tuning method that improves generator outputs using discriminator gradient signals. Chapters 4 and 5 presented finetuning methods to improve the performance of state-of-the-art pretrained GANs. In Chapter 4 we introduced a finetuning approach, PuzzleGAN, that uses a self-supervised, puzzle-solving network as an additional GAN loss term. PuzzleGAN integrates easily into existing end-to-end differentiable GAN setups and improves the global coherence of pretrained models. Chapter 5 introduced the Trained Truncation Trick (TTT), which inserts a light-weight, pre-generator module as a replacement for the test time heuristic known as the truncation trick. TTT improves output realism while better maintaining output diversity and identity than existing truncation methods.
In Chapter 6 we focused on the relatively new task of projecting real images onto the latent-noise space of StyleGAN2. Our novel projection algorithm, Fast StyleGAN2 Projection (FSP) reduces the runtime for high-quality image projection from 20 minutes to under 2 seconds. Unlike previous projection or encoder-decoder methods, FSP can project images from a wide variety of different domains.

Future work will introduce additional constraints for CoachGAN (e.g., person identity or object location) to provide more fine-grained control over output improvements. This might involve image masks to update select portions of generated output or additional loss terms to guide CoachGAN’s gradient updates in the latent space. Our experiments in Chapter 4 showed that training GANS from scratch with PuzzleGAN quickly leads to model overfit. We expect that improvements to our method will enable successful training of randomly initialized GANs and lead to further improvements in output coherence. We also hope to apply PuzzleGAN as a pretraining method for word embeddings of NLP models. Similar to the popular cloze task for NLP model pretraining (i.e. predicting a missing word in a sentence), puzzle solving can generalize to correctly ordering multiple sentences or even paragraphs in a document.

Besides inference time improvements, the Trained Truncation Trick module may prove useful as a tool to stabilize and improve GAN training. In particular, TTT could alter latent inputs to challenge both G and D during training to learn more difficult regions of the input space. Such a setup may resemble alternating least squares optimization, where TTT optimizes and then is fixed while other GAN models update.

As research continues to produce increasingly realistic and flexible GANS, we emphasize the impact of our work with real image projection. Given the lightweight parameter requirements and fast optimization times of FSP, our work could potentially reduce turnaround times for future GAN training and experimentation. We expect projection-based approaches to open doors to new inference applications, or even options for real-image, projection-guided GAN training.