# Meta Transfer GAN: A Meta-learning Approach to Few-shot GAN

Hao Zhang, Zhe Hu
{zhanghaovs120,hu.zhe}@sjtu.edu.cn

## Abstract

*Effectively training a Generative Adversarial Network (GAN) requires a large amount of data. Traditional GANs fail to adapt well to few-shot regime for over-fitting problem. Meta-learning methods are designed to address few-shot problem in classification tasks. Recently, some studies adopted meta-learning methods in training GANs in few-shot scenarios. However, typical meta-learning methods like MAML and Reptile require shallow networks which may not hold for Generative models. We propose Meta Transfer GAN, a GAN trained with Meta Transfer Learning. We show that by applying classification loss to discriminator loss, we can obtain finer features from the last convolutional layer of the discriminator. We conduct experiments in 5-way 10-shot and 3-way 10-shot tasks on two challenging data sets, miniimagenet and Fewshot-CIFAR100. Experimental results show that our method outperforms several state-of-the-art methods.*

## 1. Introduction

Generative Adversarial Networks (GANs) [6] have drawn much attention since it proposed. They lays an important role in computer vision, natural language processing, speech recognition and other fields. Despite its wide use, there are still many problems to be solved. For example, the loss of generators and discriminators cannot indicate the training process, and They lack of diversity in generating samples. Many researchers have proposed improvements to these problems in GANs. InfoGAN [14] can encode meaningful image features in a portion of the noise vector $z$ in an unsupervised manner. WGAN [1] changes the loss function to include the Wasserstein distance. Therefore, WGAN has a loss function related to image quality. In addition, training stability is improved rather than relying on architecture.But WGAN sometimes produces poor quality samples or can't converge in some settings. This is mainly due to the weighting performed in WGAN (which limits the weight of ownership to [min, max]) as a measure that satisfies the Lipschitz constraint.So WGAN-GP [7] adopts Wasserstein distance and eliminating weight clipping, so that gradient penalty can be achieved, resulting in faster convergence, higher quality samples and more stable training. WGAN-GP employs GP instead of weight-cutting training for WGAN to have faster convergence. In addition, training is more stable without hyperparameter adjustments and the architecture used is less critical. The Boundary Equilibrium GAN (BEGAN) [7] employs an automatic encoder as the GAN for the discriminator. They can successfully train through a simple architecture. They contain a dynamic item that balances the discriminator and generator during training. Although some GANs can achieve excellent performance in image classification problems, when GANs enter the one-shot, few-shot problem, training becomes very difficult and under-fitting happens frequently.

Meta-learning is widely used as a very effective way to solve the problem of few-shot. There are usually three kinds of approaches, Metric-Based, Model-Based, and Optimization-Based. Since the introduction of MAML [5], an effective solution to the problem of the few-shot learning problem has been proposed. Optimization-Based methods gain much attention in meta-learning. MAML meta-trains an initialization of the parameters and locally trains on new tasks, which leads to state-of-the-art performance on two few-shot image classification benchmarks. Reptile [9] is regarded as an improvement of MAML, which only requires first order derivation and reduced the train time tremendously. Optimization methods like MAML and Reptile work well on simple tasks with shallow networks, however, when it comes to solving complex problems with very deep network, the performance drop drastically. Meta Transfer Learning [12] extends optimization based methods to deeper network by combining meta-learning with transfer learning. It first pretrains a model, drops fully connected layers and fixes convolution layers. Then, it meta learns a new classification layer and shift and scaling (SS) parameters.

Recently, there studies working on the problem of few-shot generation. Clouatre et al. [4] combined Reptile with GAN and achieved fantastic performance. However, due to shallow network issue of Reptile, this method only works on monochrome image dataset like MINST and Omniglot. Inspired by Meta Transfer Learning, we pretrain the model on all the dataset before meta learning so that our method can be applied to far deeper networks that can deal with

more complex problems in the real world. We consider the discriminator as a classifier which can not only distinguish between real and generated data but also recognize classes of images. We also the one hot label into the generator indicting the class of the images we want to generate. Experimental results shows that our methods performs well on both monochrome images and color images like MiniImageNet. In summery, our contributions are:

- We employ meta transfer learning in training GANs to solve few-shot generation problem and our method performs better than state-of-the-art methods.
- We view the discriminator as a multi-class classifier that can recognize the class label.
- We feed class labels into the generator so that we can specify which kind of images to generate.

Few-shot generation is very useful in real life. For instance, designers and artists can create s few works and let the generation model to create beautiful art works. It can also used for data augmentation for some few-shot tasks.

## 2. Related Work

**Meta-learning:** Meta-learning, also called "learning to learn" intends to train a model which can quickly adapt to new tasks unseen previously. There are three common categories of methods. 1) Metric-based methods learn an efficient similarity or distance so that an unseen sample can be simply classified as the most similar class. 2)Memory based methods store experience learned from seen tasks and generalize the experience to unseen data. 3)Optimization based methods like MAML [5] and Reptile [9] learn a initialization of the parameters which, after training on unseen data, can fast adapt to new tasks. A serious problem of traditional meta-learning methods is that due to limited data, they usually use shallow networks, thus limiting its effectiveness. To solve this problem, Meta Transfer Learning [12] (MTL) uses pre-trained DNN and meta-learn how to effectively transfer.

**Few-Shot Generation:** Some previous studies worked on few-shot generation problem and achieved great success. Lake et al. [8] introduces a generative model of how characters are composed from strokes. It uses knowledge from previous characters to help to infer the latent strokes. Rezende et al. [10] uses a sequential generative models built on the principals of feed back and attention to achieve one shot generation. It only uses pur images and doesn't require stroke data, making this approach more general. Bartunov et al. [2] uses matching networks to achieve few-sshot generation. It adopts external memory to store learnt concepts and employs an attention model to quickly learn new concepts. It assumes that new concepts are somewhat similar to stored concepts. The fore-mentioned methods adopt Omniglot as the benchmark dataset. However, Omniglot has two main shortcomings: Its classes are all similar handwritten characters; The images lack of complexity. Furthermore, the fore-mentioned methods can only generate mono-chrome images and lack of potential to generate highly realistic images like traditional GANs do.

Clouatre et al. [4] proposed to employ Reptile to in generation tasks. Reptile [9] is a kind of first order MAML. It simply performs SGD on each task and obtain an initialization of the parameters. They do experiments on MNIST, Omniglot and a more challenging one, FIGR-8 [9], which has images from $18,409$ classes. Each class contains at least 8 images. The images are black-and-white representations of objects, concepts, patterns or designs that have been created by artists and designers. The limitations of this work are that it only works on black and white images, and the effectiveness of this method is limited by shallow network issue of Reptile.

Another work that is closely related to ours is Meta GAN [15]. Meta GAN chooses two meta learning methods, MAML and Relational Network [13], which learns distance metrics, and conbines them with GAN. The difference between Meta GAN and ours is that Meta GAN meta leans a GAN to help in few-shot classification, while our method intends to achieve few-shot generation with the help of meta learning.

## 3. Preliminary

**Generative Adversarial Network:** Goodfellow et al. [**?**] introduced the Generative Adversarial Network, a frame work training a generation model through a minmax game. The goal is to let the generated data distribution $P_G$ approximate the real data distribution $P_D$. A GAN contains two models, a generator $G$ and a discriminator $D$. In an image generation task, the discriminator tries to discriminate a generated image from a image sampled from the real data distribution, while the generator tries to "deceive" the discriminator with generated images. The generator takes a random noise $z$ as the input and output an image. The discriminator's input is an image, either from $P_G$ or $P_D$, and its output is a scalar between 0 and 1, indicating how likely the image is a real one. The minmax game is given by the following expression:

$$\begin{aligned}\min_G \max_D V(D,G) = &\mathbb{E}_{x\sim P_{\text{data}}}[\log D(x)] \\ &+\mathbb{E}_{z\sim \text{ noise}}[\log(1-D(G(z)))]\end{aligned} \tag{1}$$

**Meta Training:** For few-shot classification, we usually assume each few-shot task $\mathcal{T} \sim p(\mathcal{T})$, where $p(\mathcal{T})$ is

the distribution the tasks are sampled from. $\mathcal{T}$ includes a support set $\mathcal{T}^S$ and a query set $\mathcal{T}^Q$. Support set is used for training and query set for test. A task is called $N$-way-$K$-shot task if there are $N$ classes and $k$ samples of each class in support set. The meta training phase contains two steps, base learning and meta learning. In the base learning stage, the parameter is optimized on $\mathcal{T}^S$ according to the loss $\mathcal{L}(\theta, \mathcal{T}^S)$. The learned parameters are denoted as $\tilde{\theta}$. In the meta learning stage, the model runs on $\mathcal{T}^Q$ for each $\mathcal{T}$ with $\tilde{\theta}$ as the parameters. The losses are $\{\mathcal{L}(\theta, \mathcal{T}^Q)\}_{\mathcal{T}\sim p(\mathcal{T})}$.

**Meta Test:** In this stage, we first sample $\mathcal{T}_{test} \sim p(\mathcal{T})$. Notice that $\mathcal{T}_{test}$ and $\mathcal{T}$ are both from the same distribution, but they are disjoint. $\mathcal{T}_{test} \cap \mathcal{T} = \varnothing$. Similar to $\mathcal{T}$, we have $\mathcal{T}_{test} = \{\mathcal{T}^S_{test}, \mathcal{T}^Q_{test}\}$. The model is trained on $\mathcal{T}^S_{test}$ as in the base learning stage of the meta training phase. Finally, the trained model is evaluated on $\mathcal{T}^Q_{test}$. For generation tasks, query set of test tasks is unnecessary.

**Meta Transfer Learning:** Sun et al.[12] proposed Meta Transfer Learning. This method has the following steps.

- **Pretrain:**Train a DNN on large scale data, all available train data and discard the final layer (the fully connected layer) of the network.
- **Meta Transfer Learning (MTL):**Sample a task. Meta learn Scaling and Shifting parameters (SS) and a fully connected layer until converging. For each layer of the DNN, there is a scaling parameter $\Phi_{S_1}$ and a shifting parameter $\Phi_{S_2}$. Assuming $X$ denote the input of the layer, the weights and bias are $W$ and $b$. Then the original output of the layer is: $f(X; W, b) = WX + b$ After SS operation, the output becomes: $SS(X; W, B; \Phi_{\{1,2\}}) = (W \odot \Phi_{S_1})X + (b + \Phi_{S_2})$ Notice that the fully connected layer here is different from the one in pretrain step. This FC corresponds to a temporal classifier only working on the current task. For instance, for $N$-way few-shot tasks, the output dimension of the task is $N$.
- **Hard Tasks Learning:** Choose failure class for each task. Sample hard tasks from failure classes. Do MTL on hard tasks.

Transfer learning usually fine tunes the whole DNN when adapting to a new task. However, MTL only optimize SS parameters, keeping the DNN frozen, so that learning becomes much easier and deeper networks can be employed.

**Mode Score:** Che et al. [3] proposed Mode Score for evaluating GANs' performance, which is a improved version of Inception Score [11] (IS). Is uses an external model to evaluate the quality and diversity of images. We denote the external classifier $\mathcal{M}$. IS can be computed ass

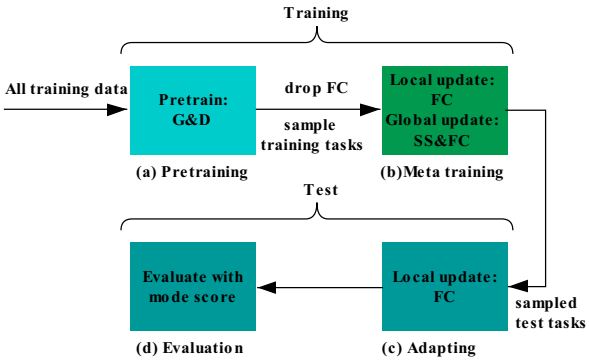


Figure 1: The pipeline of Meta Transfer GAN consists of four phases: (a)Pretrain a GAN on all train data. G denotes generator and D denotes discriminator. The first layer of the generator and the last layer of the discriminator are fully connected layers (FC), which are discarded after the pretraining phase. (b) The model learns noise mapping (NM) parameters (the first layer of G) and classifier (CF) parameters (the last layer of D) on the support set and meta learn shift and scaling (SS) parameters on the query set. (c) The model adapts to a new task by training NM and CF on the support set. (d) Evaluating the model according to mode score.

follows,

$$\mathrm{IS}\left(\mathbb{P}_g\right) = e^{\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_g}[KL(p_{\mathcal{M}}(y|\mathbf{x})\|p_{\mathcal{M}}(y))]} \tag{2}$$

where $p_{\mathcal{M}}(y|\mathbf{x})$ denotes the output distribution of $\mathcal{M}$ given input $x$. $p_{\mathcal{M}}(y)$ is the marginal of $p_{\mathcal{M}}(y|\mathbf{x})$ over the output distribution of th generator. Mode Score is a improvement version of IS. It is given by

$$\mathrm{MS}\left(\mathbb{P}_g\right) = e^{\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_g}[KL(p_{\mathcal{M}}(y|\mathbf{x})\|p_{\mathcal{M}}(y))] - KL(p_{\mathcal{M}}(y)\|p_{\mathcal{M}}(y^*))} \tag{3}$$

where $p_{\mathcal{M}}\left(y^*\right)$ is the marginal label distribution for the samples from the real data distribution. Mode Score can evaluate the dissimilarity between the real distribution and the generated distribution by $KL\left(p_{\mathcal{M}}(y)\|p_{\mathcal{M}}\left(y^*\right)\right)$.

## 4. Our Model

As shown in Figure 1, our method includes 4 phases. First, we pretrain a GAN on all the images in training set. Then we drop the fully connected layers of the generator and the discriminator and fix the remnant layers as image constructor (IC) and feature extractor (FE). The next is the meta training phase. We learn NM and CF on the support set of each task and meta learn SS on query sets. When testing, we first train NM and CF on the support set of each test task. Finally, the model is evaluated with mode score metric.

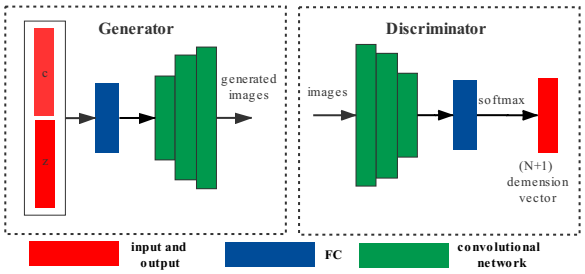


Figure 2: The structure of our proposed model consists of a generator and a discriminator like a regular GAN. The generator takes concatenated random noise $z$ and one-hot label $c$ as input. The first layer is the noise mapping layer which maps the input into a feature. Then the image constructor convert the feature vector into a image. The discriminator first extracts features from input images and then classify it through a fully connected layer. The output is activated using softmax.

### 4.1. Pretraining

We first train a GAN on large scale data set. For fair comparison, we do not train the model on Image Net like typical transfer learning does. In stead, we pretrain on all the training accessible data. Of the proposed GAN, the first layer of the generator and the last layer of the discriminator are fully connected layers as in Figure 2. We call them noise mapping (NM) layer and classification (CF) layer respectively. Different from traditional GAN, the discriminator here is a multi-class classifier, the output of which is not a scalar but a vector. For instance, for miniimagenet which has totally 64 classes of images in the training set, the output dimension of the discriminator is 65. As the output is activate with softmax, each dimension can represent the probability that the image belongs to the corresponding class. The additional one dimension represents the probability of the image being a fake (generated) image. In addition, our proposed GAN takes as input not only a random noise $z$ but also a obe-hot class indicator $c$. To be more specific, assuming we have $N$-way-$K$-shot few-shot image generation tasks. $x$ is an image, $y$ is the class indicator of $x$, and $1 \leq y \leq N$. $c = OneHot(y)$. If $x$ is a real image, $y$ is the class $x$ belongs to, while if $x$ is a generated image, $y$ means we want to generate an image $x$ that seems to belong to $y$. The optimization target of the discriminator is to classify images correctly. The output of $D$ is $D(x) = (D_1(x), D_2(x), ..., D_{N+1}(x))$. The optimization target of $D$ is as follows:

$$\begin{aligned}\max_D V_D(D,G) = E_{x\sim P_{data}}\left[\log\left(D_y(x)\right)\right] \\ +E_{z\sim noise}\left[\log\left(D_{N+1}(G(z,y))\right)\right]\end{aligned} \tag{4}$$

One should notice that $D_{N+1}(x) = 1 - \sum_{i=1}^{N} D_i$. So when $N = 1$ Equation (1) and (4) are the same. The goal of the generator is to deceive the discriminator, let the generated image be classified as any class except $N + 1$.

$$\min_G V_G(D,G) = E_{z\sim noise}\left[\log\left(\sum_{i=1}^{N} D_i(G(z,y))\right)\right] \tag{5}$$

Actually, the generator should have a higher goal, to let the generated image to be classified "correctly", that is, to be $y$.

$$\min_G V_G(D,G) = E_{z\sim noise}\left[\log\left(D_y(G(z,y))\right)\right] \tag{6}$$

We adopt Equation (6) as the optimization goal of the generator.

Given the optimization targets, we can obtain the loss functions of D and G. We first sample a batch of $m$ real images and corresponding labels $R = \{(x_1,y_1),(x_2,y_2),...,(x_m,y_1)\}$ and $n$ generated images and their labels $F = \{(G(z_1),y_1),(G(z_2),y_2),...,(G(z_n),y_n)\}$. The loss functions of G and D is as follows.

$$\begin{aligned}L_D(R,F) = &-\sum_{x,y\in R}\log\left(D_y(x)\right) \\ &-\sum_{x,y\in F}\log\left(1 - D_{N+1}(x)\right) \\ L_G(R,F) = &-\sum_{x,y\in F}\log\left(D_y(x)\right)\end{aligned} \tag{7}$$

Denoting the parameters of convolution layers as $\Theta = \{\Theta_G, \Theta_D\}$, we update $\Theta$ as follows.

$$\widetilde{\Theta}_G = \Theta_G - \lambda\nabla_{\Theta_G} L_G(R,F) \tag{8}$$

$$\widetilde{\Theta}_D = \Theta_D - \lambda\nabla_{\Theta_D} L_G(R,F) \tag{9}$$

Once pretraining is finished, FCs are dropped and parameters in IC and FE will be fixed in following phases.

### 4.2. Meta Training

Meta Training contains two steps. All the parameters need to update in meta traing is denoted as $\Phi = \{\Phi_{NM}, \Phi_{CF}, \Phi_{SS}\}$ In the first step, we update the fully connected layers NM and CF on the support set. The loss function are the same as in pretraining. $\Phi^i_{NM}$ and $\Phi^i_{CP}$ denote the updated parameters on task $i$.

$$\begin{aligned}\Phi^i_{NM} &= \Phi_{NM} - \lambda\nabla_{\Phi_{MM}} L_G\left(R^i, F^i; \Phi\right) \\ \Phi^i_{CF} &= \Phi_{CF} - \lambda\nabla_{\Phi_{CF}} L_D\left(R^i, F^i; \Phi\right) \\ &R^i, F^i \in T^i\end{aligned} \tag{10}$$

$\lambda$ is the learning rate.

When the above update finishes, we can meta learn SS with learned NM and CF. $\Phi_{SS}$ denotes shift and scaling parameters. Different from the first step, we do the following update on the query sets $\mathcal{T}^Q$.

$$\widetilde{\Phi}_{SS} = \Phi_{SS} - \lambda \nabla_{\Phi_{SS}} \sum_i L\left(R^i, F^i; \Phi^i\right) \qquad (11)$$

$\Phi^i = \{\Phi^i_{NM}, \Phi^i_{CP}, \Phi_{SS}\}$ We use the same notation $\lambda$ to denote the learning rate, but the learning rate do not have to be the same as in Equation (10). After convergence, we obtain $\widetilde{\Phi} = \{\widetilde{\Phi}_{NM}, \widetilde{\Phi}_{CF}, \widetilde{\Phi}_{SS}\}$, where $\widetilde{\Phi}_{NM}$ and $\widetilde{\Phi}_{CF}$ are the averages of all the $\Phi^i_{NM}$'s and $\Phi^i_{CF}$'s respectively.

$$\widetilde{\Phi}_{NM} = \frac{1}{T}\sum_i \Phi^i_{NM}, \widetilde{\Phi}_{CF} = \frac{1}{T}\sum_i \Phi^i_{CF} \qquad (12)$$

**Shift and Scaling (SS):** SS is a series of parameters $\{\Phi^1_{S_{1,2}}, \Phi^2_{S_{1,2}}, ...\Phi^L_{S_{1,2}}, \}$ for an $L$-layer convolution network. $Phi^l_{S_1}$ and $Phi^l_{S_2}$ are both scalars. They are called scaling and shift parameters respectively. If we note the input, weights and bias of the $l$th layer as $X^l, W^l$ and $b^l$, the output of the layer should be $W^l X^l + b^l$. With SS, the output becomes $\Phi^l_1 W^l X^l + b^l + \Phi^l_2$. Unlike fine tuning that need to update the complete values of $W^l$ and $b^l$, which has a large amount of parameters, SS can reduce this number dramatically (to 2/9 as Sun et al. [12] did).

### 4.3. Test

Given a test task $\mathcal{T}_{test} = \{\mathcal{T}^S_{test}\}$. We update NM and CF as in Equation (10). As we do not need to meta learn in this phase, we do not need a query set. Finally, we can calculate mode score with Equation 3.

Algorithm 1 summarize the training processes.

**Algorithm 1** Meta Transfer GAN

**Input:** $p(\mathcal{T})$
**Output:** $\widetilde{\Theta}$, $\widetilde{\Phi}$
Initialize $\Phi_{NM}$, $\Phi_{CF}$, $\Theta$ randomly.
**while** $NotConverge$ **do**
  Update $\Theta$ into $\widetilde{\Theta}$ according to Equation (8) and (9).
**end while**
Initialize $\Phi_{S_1}$ as 1s and $\Phi_{S_2}$ as 0s.
**while** $NotConverge$ **do**
  Sample a batch of $T$ tasks $ST$ from $p((T))$
  **for** $\mathcal{T}^i \in ST$ **do**
    **while** $NotConverge$ **do**
      Sample $R^i$ and $F^i$ from $\mathcal{T}^i$
      Update $\Phi$ into $\Phi_i$ through Equation (10)
    **end while**
  **end for**
  Calculate $\widetilde{\Phi}_{NM}$ and $\widetilde{\Phi}_{CF}$ by Equation (12)
  Update $\Phi_{SS}$ into $\widetilde{\Phi_{SS}}$ by Equation (11)
**end while**

## References


[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. 2017.

[2] Sergey Bartunov and Dmitry P Vetrov. Few-shot Generative Modelling with Generative Matching Networks. Technical report, 2018.

[3] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. MODE REGULARIZED GENERATIVE ADVERSARIAL NETWORKS. Technical report.

[4] Louis Clouâtre and Marc Demers. FIGR: Few-shot Image Generation with Reptile. 2019.

[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *34th International Conference on Machine Learning, ICML 2017*, 3:1856–1868, 2017.

[6] Ian Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks. 2016.

[7] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs Montreal Institute for Learning Algorithms. Technical report.

[8] Brenden M Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B Tenenbaum. One shot learning of simple visual concepts. Technical report.

[9] Alex Nichol, Joshua Achiam, and John Schulman Openai. On First-Order Meta-Learning Algorithms. Technical report.

[10] Danilo J Rezende, Shakir Mohamed, Karol Gregor, and Daan Wierstra. One-Shot Generalization in Deep Generative Models Ivo Danihelka. Technical report, 2016.

[11] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[12] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-Transfer Learning for Few-Shot Learning. Technical report.

[13] Flood Sung Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to Compare: Relation Network for Few-Shot Learning. Technical report.

[14] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-Attention Generative Adversarial Networks. 2018.

[15] Ruixiang Zhang, Tong Che, Zoubin Grahahramani, Yoshua Bengio, and Yangqiu Song. MetaGAN: An Adversarial Approach to Few-Shot Learning. Technical report.