==

Final Report——Few-shot Generation

CS7335-033-M01-Statistical Learning and Inference, Li Niu, Autumn 2020.

∗ Name: Jiadong Chen Student ID: 120033910065 Email: 2235372520@qq.com
∗ Name: Haoyi Hu Student ID: 120033910071 Email: 565287421@qq.com
∗ Name:Runfeng Chen Student ID:120033910034 Email: 874487625@qq.com

Contents

1 Introduction **2**

2 Related Work **2**
2.1 Meta Learning 2
2.1.1 Overview 2
2.1.2 Formalizing Meta-learning from Different Perspectives 2
2.2 Few-shot Learning 4
2.2.1 Overview 4
2.2.2 Different Methods for Few-Shot Learning 4
2.3 Advanced Models for Few-Shot Generation 5
2.3.1 Data Augmentation Generative Adversarial Networks(DAGAN) 5
2.3.2 Fusing-and-Filling GAN for Few-shot Image Generation(F2GAN) 6

3 Method **6**
3.1 Similarity DAGAN 6
3.2 Class-aware Data Augmentation Generative Adversarial Networks 7
3.3 Distribution influenced GAN 9

4 Experimental Setting **10**
4.1 Datasets and Implentation Details 10
4.2 Generated Images 10

1 Introduction

Although the current generation model performs well, it often requires a large amount of training data, and its performance will decline rapidly when the amount of data is small. However, for emerging categories or long-tail categories, it is very difficult and expensive to obtain data, so we need to expand the data. Then, new technology named as few-shot generation is discovered.The applications of few-shot image generation are broad. It can benefit a wide range of downstream category-aware tasks like few-shot classification.

There has been a lot of few-shot image generation models which perform well, such as MathcingGAN, F2GAN, DAGAN e.t.c. Here, we made some modifications to DAGAN in order to get a more diverse picture.

2 Related Work

2.1 Meta Learning

2.1.1 Overview

Meta-learning is most commonly understood as *learning to learn*, which refers to the process of improving a learning algorithm over multiple learning episodes. In contrast, conventional ML improves model predictions over multiple data instances. During **base learning**, an *inner* (or *lower/base*) learning algorithm solves a task such as image classification, defined by a dataset and objective. During **meta-learning**, an *outer* (or *upper/meta*) algorithm updates the inner learning algorithm such that the model it learns improves an outer objective. For instance this objective could be generalization performance or learning speed of the inner algorithm. Learning episodes of the base task, namely (base algorithm, trained model, performance) tuples, can be seen as providing the instances needed by the outer algorithm to learn the base learning algorithm. As defined above, many conventional algorithms such as random search of hyper-parameters by cross-validation could fall within the definition of meta-learning. The salient characteristic of contemporary neural-network metalearning is an explicitly defined meta-level objective, and end to-end optimization of the inner algorithm with respect to this objective.

2.1.2 Formalizing Meta-learning from Different Perspectives

Conventional Machine Learning In conventional supervised machine learning, we are given a training dataset $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, such as (input image, output label) pairs. We can train a predictive model $\hat{y} = f_\theta(x)$ parameterized by θ, by solving:

$$\theta^* = \arg\min_\theta \mathcal{L}(\mathcal{D}; \theta, \omega) \tag{1}$$

where $\mathcal{L}$ is a loss function that measures the error between true labels and those predicted by $f_\theta(\cdot)$. The conditioning on ω denotes the dependence of this solution on assumptions about 'how to learn', such as the choice of optimizer for θ or function class for f. Generalization is then measured by evaluating a number of test points with known labels. The conventional assumption is that this optimization is performed from scratch for every problem $\mathcal{D}$; and that ω is pre-specified. However, the specification of ω can drastically affect performance measures like accuracy or data efficiency. Meta-learning seeks to improve these measures by learning the learning algorithm itself, rather than assuming it is prespecified and fixed. This is often achieved by revisiting the first assumption above, and learning from a distribution of tasks rather than from scratch.

Meta-Learning: Task-Distribution View A common view of meta-learning is to learn a general purpose learning algorithm that can generalize across tasks, and ideally enable each new task to be learned better than the last. We can evaluate the performance of ω over a distribution of tasks $p(\mathcal{T})$. Here we loosely define a task to be a dataset and loss function $\mathcal{T}=\{\mathcal{D},\mathcal{L}\}$. Learning how to learn thus becomes

$$\min_{\omega} \underset{\mathcal{T}\sim p(\mathcal{T})}{\mathbb{E}} \mathcal{L}(\mathcal{D};\omega) \tag{2}$$

where $\mathcal{L}(\mathcal{D};\omega)$ measures the performance of a model trained using ω on dataset $\mathcal{D}$. 'How to learn', i.e. ω, is often referred to as across-task knowledge or meta-knowledge.

To solve this problem in practice, we often assume access to a set of source tasks sampled from $p(\mathcal{T})$. Formally, we denote the set of M source tasks used in the meta-training stage as $\mathscr{D}_{\text{source}}=\left\{\left(\mathcal{D}_{\text{source}}^{\text{train}},\mathcal{D}_{\text{source}}^{\text{val}}\right)^{(i)}\right\}_{i=1}^{M}$ where each task has both training and validation data. Often, the source train and validation datasets are respectively called support and query sets. The meta-training step of 'learning how to learn' can be written as:

$$\omega^{*}=\arg\max_{\omega}\log p\left(\omega \mid \mathscr{D}_{\text{source}}\right) \tag{3}$$

Now we denote the set of Q target tasks used in the meta-testing stage as $\mathscr{D}_{\text{target}}=\left\{\left(\mathcal{D}_{\text{target}}^{\text{train}},\mathcal{D}_{\text{target}}^{\text{test}}\right)^{(i)}\right\}_{i=1}^{Q}$ where each task has both training and test data. In the metatesting stage we use the learned meta-knowledge ω^{*} to train the base model on each previously unseen target task i :

$$\theta^{*(i)}=\arg\max_{\theta}\log p\left(\theta \mid \omega^{*},\mathcal{D}_{\text{target}}^{\text{train}(i)}\right) \tag{4}$$

In contrast to conventional learning in Eq. 1 , learning on the training set of a target task i now benefits from meta-knowledge ω^{*} about the algorithm to use. This could be an estimate of the initial parameters, or an entire learning model or optimization strategy. We can evaluate the accuracy of our meta-learner by the performance of $\theta^{*}(i)$ on the test split of each target task $\mathcal{D}_{\text{target}}^{\text{test}(i)}$ This setup leads to analogies of conventional under-fitting and over-fitting: meta-under fitting and meta-over-fitting. In particular, meta-over fitting is an issue whereby the meta knowledge learned on the source tasks does not generalize to the target tasks. It is relatively common, especially in the case where only a small number of source tasks are available. It can be seen as learning an inductive bias ω that constrains the hypothesis space of θ too tightly around solutions to the source tasks.

Meta-Learning: Bi-level Optimization View The previous discussion outlines the common flow of meta-learning in a multiple task scenario, but does not specify how to solve the meta-training step in Eq. 3. This is commonly done by casting the meta-training step as a bi-level optimization problem. While this picture is arguably only accurate for the optimizer-based methods, it is helpful to visualize the mechanics of meta-learning more generally. Bi-level optimization refers to a hierarchical optimization problem, where one optimization contains another optimization as a constraint. Using this notation, meta-training can be formalised as follows:

$$\omega^{*}=\arg\min_{\omega}\sum_{i=1}^{M}\mathcal{L}^{\text{meta}}\left(\theta^{*(i)}(\omega),\omega,\mathcal{D}_{\text{source}}^{\text{val}(i)}\right) \tag{5}$$

$$\text{s.t. } \theta^{*(i)}(\omega)=\arg\min_{\theta}\mathcal{L}^{\text{task}}\left(\theta,\omega,\mathcal{D}_{\text{source}}^{\text{train }(i)}\right) \tag{6}$$

where $\mathcal{L}^{\text{meta}}$ and $\mathcal{L}^{\text{tank}}$ refer to the outer and inner objectives respectively, such as cross entropy in the case of few-shot classification. Note the leader-follower asymmetry between the outer and inner levels: the inner level optimization Eq. 6 is conditional on the learning strategy ω defined by the outer level, but it cannot change ω during its training. Here ω could indicate an initial condition in non-convex optimization, a hyper-parameter such as regularization strength or even a parameterization of the loss function to optimize $\mathcal{L}^{\text{tank}}$.

2.2 Few-shot Learning

2.2.1 Overview

In order to learn from a limited number of examples with supervised information, a new machine learning paradigm called Few-Shot Learning (FSL) is proposed. A typical example is character generation, in which computer programs are asked to parse and generate new handwritten characters given a few examples. To handle this task, one can decompose the characters into smaller parts transferable across characters, and then aggregate these smaller components into new characters. This is a way of learning like human. Naturally, FSL can also advance robotics, which develops machines that can replicate human actions. Examples include one-shot imitation, multi-armed bandits, visual navigation, and continuous control.

Formally, Few-Shot Learning (FSL) is a type of machine learning problems (specified by from experience E with respect to some classes of task T and performance measure P), where E contains only a limited number of examples with supervised information for the target T.

2.2.2 Different Methods for Few-Shot Learning

In order to approximate the ground-truth hypothesis $\hat{h}$, the model has to determine a hypothesis space $\mathcal{H}$ containing a family of hypotheses h 's, such that the distance between the optimal $h^* \in \mathcal{H}$ and $\hat{h}$ is small.

Given the few-shot dataset D_{train} with limited samples, one can choose a small $\mathcal{H}$ with only simple models (such as linear classifiers). However, real-world problems are typically complicated, and cannot be well represented by an hypothesis h from a small $\mathcal{H}$. Therefore, a large enough $\mathcal{H}$ is preferred in FSL, which makes standard machine learning models infeasible. FSL methods manage to learn by constraining $\mathcal{H}$ to a smaller hypothesis space $\tilde{\mathcal{H}}$ via prior knowledge in E. The empirical risk minimizer is then more reliable, and the risk of overfitting is reduced.

Multitask Learning In the presence of multiple related tasks, multitask learning learns these tasks simultaneously by exploiting both task-generic and task-specific information. Hence, they can be naturally used for FSL. Here, we present some instantiations of using multitask learning in FSL.

We are given C related tasks $T_1, \ldots, T_C$, in which some of them have very few samples while some have a larger number of samples. Each task T_c has a data set $D_c = \{D^c_{\text{train}}, D^c_{\text{test}}\}$, in which D^c_{train} is the training set and D^c_{test} is the test set. Among these C tasks, we regard the few-shot tasks as target tasks, and the rest as source tasks. Multitask learning learns from D^c_{train} 's to obtain θ_c for each T_c. As these tasks are jointly learned, the parameter θ_c of h_c learned for task T_c is constrained by the other tasks. According to how the task parameters are constrained, we can divide methods in this strategy as parameter sharing; and parameter tying.

Embedding Learning Embedding learning embeds each sample $x_i \in X \subseteq \mathbb{R}^d$ to a lower-dimensional $z_i \in \mathcal{Z} \subseteq \mathbb{R}^m$, such that similar samples are close together while dissimilar samples can be more easily differentiated. In this lower-dimensional $\mathcal{Z}$, one can then construct a smaller hypothesis space $\mathcal{H}$ which subsequently requires fewer training samples. The embedding function is mainly learned from prior knowledge, and can additionally use task-specific information from $D_{\text{train.}}$

Embedding learning has the following key components: (i) a function f which embeds test sample $x_{\text{test}} \in D_{\text{test}}$ to $\mathcal{Z}$, (ii) a function g which embeds training sample $x_i \in D_{\text{train}}$ to $\mathcal{Z}$, and (iii) a similarity function $s(\cdot, \cdot)$ which measures the similarity between $f(x_{\text{test}})$ and $g(x_i)$ in $\mathcal{Z}$. The test sample x_{test} is assigned to the class of x_i, whose embedding $g(x_i)$ is most similar to $f(x_{\text{test}})$ in $\mathcal{Z}$ according to s. Although one can use a common embedding function for both x_i and x_{test}, using two separate embedding functions may obtain better accuracy.

According to whether the parameters of embedding functions f and g vary across tasks, we classify these FSL methods as using a task-specific embedding model; task-invariant (i.e., general) embedding model; and hybrid embedding model, which encodes both task-specific and task-invariant information.

Learning with External Memory Learning with external memory extracts knowledge from $D_{\text{train, and stores it in}}$ an external memory (Figure 9). Each new sample x_{test} is then represented by a weighted average of contents extracted from the memory. This limits x_{test} to be represented by contents in the memory, and thus essentially reduces the size of $\mathcal{H}$.

A key-value memory is usually used in FSL. Let the memory be $M \in \mathbb{R}^{b \times m}$, with each of its b memory slots $M(i) \in \mathbb{R}^m$ consisting of a key-value pair $M(i) = (M_{\text{key}}(i), M_{\text{value}}(i))$. A test sample x_{test} is first embedded by an embedding function f. However, unlike embedding methods, $f(x_{\text{test}})$ is not used directly as the representation of $x_{\text{test.}}$ Instead, it is only used to query for the most similar memory slots, based on the similarity $s(f(x_{\text{test}}), M_{\text{key}}(i))$ between $f(x_{\text{test}})$ and each key $M_{\text{key}}(i)$. The values of the most similar memory slots ($M_{\text{value}}(i)'$s) are extracted and combined to form the representation of $x_{\text{test.}}$ This is then used as input to a simple classifier (such as a softmax function) to make prediction. As manipulating M is expensive, M usually has a small size. When M is not full, new samples can be written to vacant memory slots. When M is full, one has to decide which memory slots to be replaced.

Generative Modeling Generative modeling methods estimate the probability distribution $p(x)$ from the observed x_i 's with the help of prior knowledge. Estimation of $p(x)$ usually involves estimations of $p(x \mid y)$ and $p(y)$. Methods in this class can deal with many tasks, such as generation recognition, reconstruction, and image flipping.

In generative modeling, the observed x is assumed to be drawn from some distribution $p(x; \theta)$ parameterized by θ. Usually, there exists a latent variable $z \sim p(z; \gamma)$, so that $x \sim \int p(x \mid z; \theta) p(z; \gamma) dz$. The prior distribution $p(z; \gamma)$, which is learned from other data sets, brings in prior knowledge that is vital to FSL. By combining the provided training set D_{train} with this $p(z; \gamma)$, the resultant posterior probability distribution is constrained. In other words, $\mathcal{H}$ is constrained to a much smaller $\tilde{\mathcal{H}}$.

2.3 Advanced Models for Few-Shot Generation

2.3.1 Data Augmentation Generative Adversarial Networks(DAGAN)

DAGAN takes advantages of both Data Augmentation and Generative Adversarial Networks to finish the Few-shot generation work.

Data Augmentation is routinely used in classification problems. Often it is non-trivial to encode known invariances in a model. It can be easier to encode those invariances in the data instead by generating additional data items through transformations from existing data items. For example the labels of handwritten characters should be invariant to small shifts in location, small rotations or shears, changes in intensity, changes in stroke thickness, changes in size etc. Almost all ncases of data augmentation are from a priori known invariance.

Generative Adversarial Networks(GAN) is a class of machine learning frameworks with two neural networks contest with each other in a game (in the form of a zero-sum game, where one agent's gain is another agent's loss). Given a training set, this technique learns to generate new data with the same statistics as the training set. For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics. Though originally proposed as a form of generative model for unsupervised learning, GANs have also proven useful for semi-supervised learning, fully supervised learning, and reinforcement learning. The core idea of a GANs is based on the "indirect" training through the discriminator, which itself is also being updated dynamically. This basically means that the generator is not trained to minimize the distance to a specific image, but rather to fool the discriminator. This enables the model to learn in an unsupervised manner.

As shown in Fig.**??**, the DAGAN is composed of two parts. Left part, the generator network is composed of an encoder taking an input image (from class c), projecting it down to a lower dimensional manifold (bottleneck). A random vector (z_i) is transformed and concatenated with the bottleneck vector; these are both passed to the decoder network which generates an augmentation image. the right part, the adversarial discriminator network is trained to discriminate between the samples from the real distribution (other real images from the same class) and the fake distribution (images generative from the generator network). Adversarial training leads the network to generate new images from an old one that appear to be within the same class (whatever that class is), but look different enough to be a different sample.

Data augmentation is a widely applicable approach to improving performance in low-data setting, and a DAGAN is a flexible model to automatic learn to augment data. However beyond that, the authors demonstrate that DAGAN improve performance of classifiers even after standard data-augmentation. Furthermore by meta-learning the best choice of augmentation in a one-shot setting it leads to better performance than other state of the art meta learning methods. The generality of data augmentation across all models and methods means that a DAGAN could be a valuable addition to any low data setting.

2.3.2 Fusing-and-Filling GAN for Few-shot Image Generation(F2GAN)

Fusing-and-Filling GAN for Few-shot Image Generation(F2GAN) is one of the **state-of-the-art** modles to generate realistic and diverse images for a new category with only a few images. In F2GAN, the authors designed a fusion generator that fuses high-level features of conditional images with random interpolation coefficients, and then fills low-level details with a non-local attention fusion module to generate a new image.In addition, the discriminator ensures the diversity of generated images by looking for pattern loss and interpolating regression loss.

3 Method

3.1 Similarity DAGAN

Data Augmentation Generative Adversarial Networks(DAGAN) has a good idea that design a unique discriminator that takes

1. some input data point x_i and a second data point from the same class: x_j such that $t_i = t_j$
2. some input data point x_i and the output of the current generator x_g which takes x_i as an input.

The critic tries to discriminate the generated points (b) from the real points (a). The generator is trained to minimize this discriminative capability as measured by the Wasserstein distance.

The importance of providing the original x to the discriminator should be emphasised. They want to ensure the generator is capable of generating different data that is related to, but different from, the current data point. By providing information about the current data point to the discriminator they prevent the GAN from simply autoencoding the current data point. At the same time we do not provide class information, so it has to learn to generalise in ways that are consistent across all classes.

However, the lack of diversity still exists in the image generated by DAGAN because it just let the generated image be as likely as the second data point x_j. So, we add a new parameter c to control the similarity between the generated points (b) from the real points (a). Then we can generate the image between x_i and x_j.

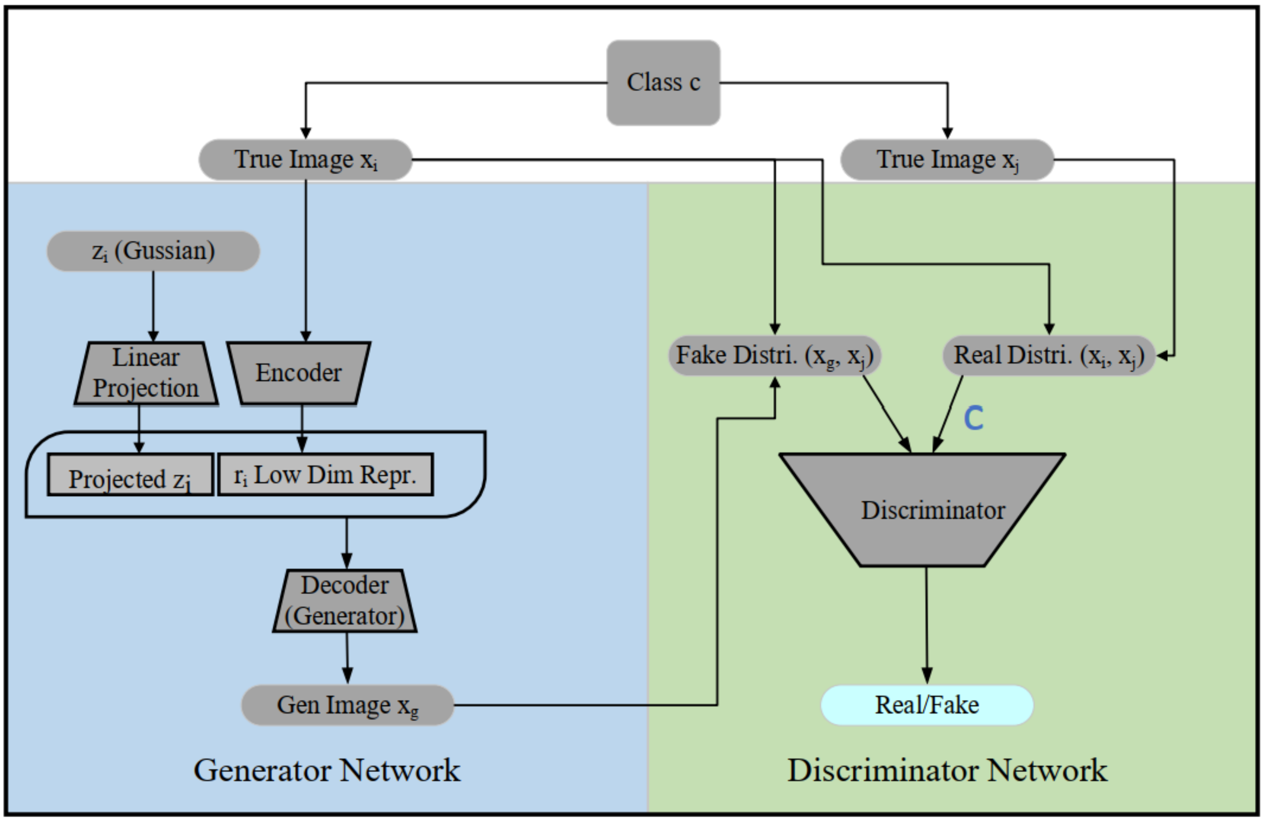

Figure 1: Similarity DAGAN

3.2 Class-aware Data Augmentation Generative Adversarial Networks

Data Augmentation Generative Adversarial Networks(DAGAN) has a intuitive idea that use an Gaussian noise to make the generated images diverse, the related formula is described as:

$$\begin{aligned}\mathbf{z} &= \widetilde{N}(\mathbf{0},\ \mathbf{I})\\ \mathbf{r} &= g(\mathbf{x}_{true})\\ \mathbf{x}_{gen} &= f(\mathbf{z},\ \mathbf{r})\end{aligned} \tag{7}$$

where $\mathbf{x}_{true}$ is a given image, $\mathbf{x}_{gen}$ are the vectors being generated (that, in distribution, should match the data D), $\mathbf{z}$ are the latent Gaussian variables that provide the variation in what is generated, g is the Encoder(implemented via a neural network), and f is the Decoder(also implemented via a neural network).

While DAGAN performs well on some datasets, it takes a lot of time to train the Encoder and Decoder on some large-scale data sets, so we develop a model, called Class-aware Data Augmentation Generative Adversarial Networks(ClassDAGAN), in order to make the networks converge faster, whose structure is shown in Fig.5.

The related formula can be described as:

$$\begin{aligned}\mathbf{c} &= e(\mathbf{C}), \mathbf{C} = \{x_i | x_i\ \textit{has the same class lable with}\ \mathbf{x}_{ture}\}\\ \mathbf{z} &= \widetilde{N}(\mathbf{0},\ \mathbf{I})\\ \mathbf{r} &= g(\mathbf{x}_{true})\\ \mathbf{x}_{gen} &= f(\mathbf{z},\ \mathbf{r},\ \mathbf{c})\end{aligned} \tag{8}$$

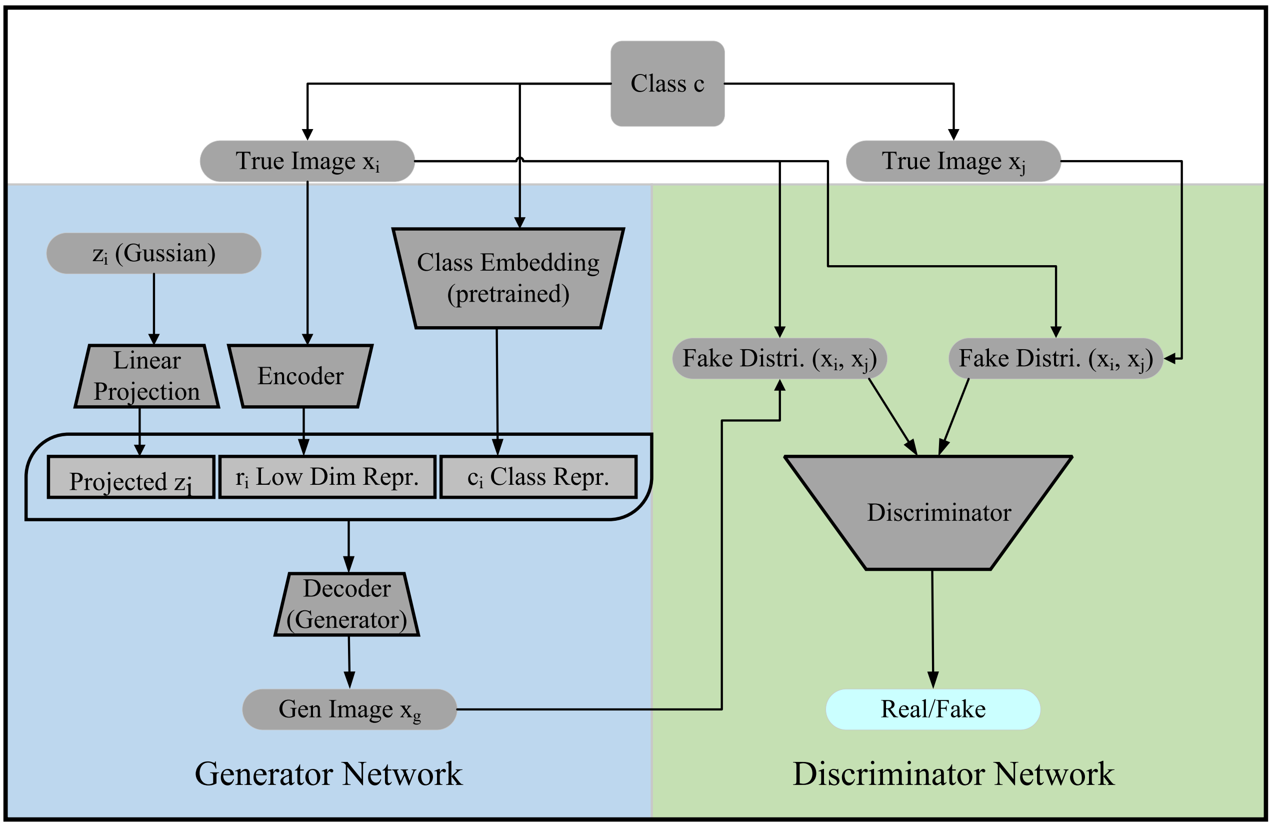

Figure 2: The architecture of Class-aware Data Augmentation Generative Adversarial Networks

of which e is an pre-trained embedding model to embed the class into a vector c, and the other notations hold the same definitions with original DAGAN.

In practice we design different embedding methods, such as Auto-Encoder, and Attention-based Matching Network. For Auto-Encoder, we use some source domains consisting of data $D^i = \left\{x_1^i, x_2^i, \cdots, x_{n_{D^i}}^i\right\}$ $(1 \leq i \leq N)$ to train an Auto-Encoder, and for an image x, the embedded vecotr c is the output of this Auto-Encoder with input $\overline{x} = \frac{1}{n_{D^i}} \sum_{i=1}^{n_{D^i}} x_i^i$.

The data augmentation model can be learnt in the source domain using an adversarial approach. Consider a source domain consisting of data $D = \{\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_{N_D}\}$ and corresponding target values $\{t_1, t_2, \ldots, t_{N_D}\}$. We use an improved WGAN, critic that either takes

1. some input data point x_i and a second data point from the same class: x_j such that $t_i = t_j$
2. some input data point x_i and the output of the current generator x_g which takes x_i as an input.

The critic tries to discriminate the generated points (b) from the real points (a). The generator is trained to minimize this discriminative capability as measured by the Wasserstein distance.

The importance of providing the original x to the discriminator should be emphasised. We want to ensure the generator is capable of generating different data that is related to, but different from, the current data point. By providing information about the current data point to the discriminator we prevent the GAN from simply autoencoding the current data point. At the same time we do not provide class information, so it has to learn to generalise in ways that are consistent across all classes.

3.3 Distribution influenced GAN

Generative Adversarial Methods are the most widely used methods for learning to generate examples from density.Usually they learn a generator and a discriminator by minimizing a distribution measure between the generated image and the true image. And the generative model is learnt by a GAN network.

Usually, the data that generator need have two parts, noise and true image. Noise is usually a Gaussian noise, it is used to make the image generated more variant.

$$\mathbf{z} = \widetilde{N}(\mathbf{0}, \mathbf{I}) \tag{9}$$

z is the Gaussian noise, mean is 0 and variance is 1.

$$l = L(z) \tag{10}$$

L is a linear function, and l are the vectors generated. l will be injected into the auto encoder to provide the variation.

But how to control the variation is a question. Sometimes the variation is too large and the change in generated images are too big that it is very hard for the generator to generate good images. If the variation is too small, then the generator cannot generate enough different pictures.So we want to control the variant of the images.

Suppose there are N images in one batch. First we calculate the distribution on each pixel and get $\mathbb{P}$.$\mathbb{P}$ is a distribution matrix with dimension $h * w$. Each element represents a distribution in this pixel. Then we generate N random Gaussian matrix and calculate the distribution on each pixel just like $\mathbb{P}$ and get $\mathbb{Q}$. Then we enlarge the interval of the minimum and maximum value in $\mathbb{Q}$ into [0,256) so that it is consist with $\mathbb{P}$. Then quantum each value and let them be one integer between zero and 256. To calculate the distance between two distributions, we use KL divergence.

$$D_{KL}(p||q) = \sum_{i=1}^{N} p_i log \frac{p_i}{q_i} \tag{11}$$

$\mathbb{P}$ represents the true distribution of images in one batch and $\mathbb{Q}$ is noises. The more similar the two distribution are, the smaller the value of KL divergence is. If $\mathbb{P}$ and $\mathbb{Q}$ are totally equal, the KL divergence $\mathbb{D}$ is 0. To transform the distance into a coefficient, we need a function that can projection the interval (0,+∞) into (0,1). There are many function can achieve this, we choose

$$f(x) = e^{-x} \tag{12}$$

$$e = f(D) \tag{13}$$

D is the KL divergence and e is the coefficient. We use α as the hyperparameter.It is set by human representing the intent that people are willing to see different pictures. We use the mean of the N Gaussian vectors as the injected noise.

$$q = \sum_{i=1}^{N} \mathbb{Q}_i \tag{14}$$

At last, the vector to be injected is:

$$z = q * max(\alpha, e) \tag{15}$$

If you want to see very different pictures(but at the same time the pictures maybe nonsense) you can set α high. If you want to generate only very similar but meaningful pictures, you can set α low.

4 Experimental Setting

4.1 Datasets and Implentation Details

Although we designed three methods, we experimented with only the first method SDAGAN due to time and computational resources. We tested the SDAGAN augmentation on Omniglot dataset.

For classifier networks, all data for each character (handwritten or person) was further split into 2 test cases (for all datasets), 3 validation cases and a varying number of training cases depending on the experiment. Classifier training was done on the training cases for all examples in all domains, with hyperparameter choice made on the validation cases. Finally test performance was reported only on the test cases for the target domain set. Case splits were randomized across each test run.

For one-shot networks, DAGAN training was done on the source domain, and the meta learning done on the source domain, and validated on the validation domain. Results were presented on the target domain data. Again in the target domain a varying number of training cases were provided and results were presented on the test cases (2 cases for each target domain class in all datasets).

The Omniglot data (Lake et al., 2015) was split into source domain and target domain similarly to the split in (Vinyals et al., 2016). The order of the classes was shuffled such that the source and target domains contain diverse samples (i.e. from various alphabets). The first 1200 were used as a source domain set, 1201-1412 as a validation domain set and 1412-1623 as a target domain test set.

4.2 Generated Images

These are the generated images using our model. We selected some well recognized pictures and put it below.

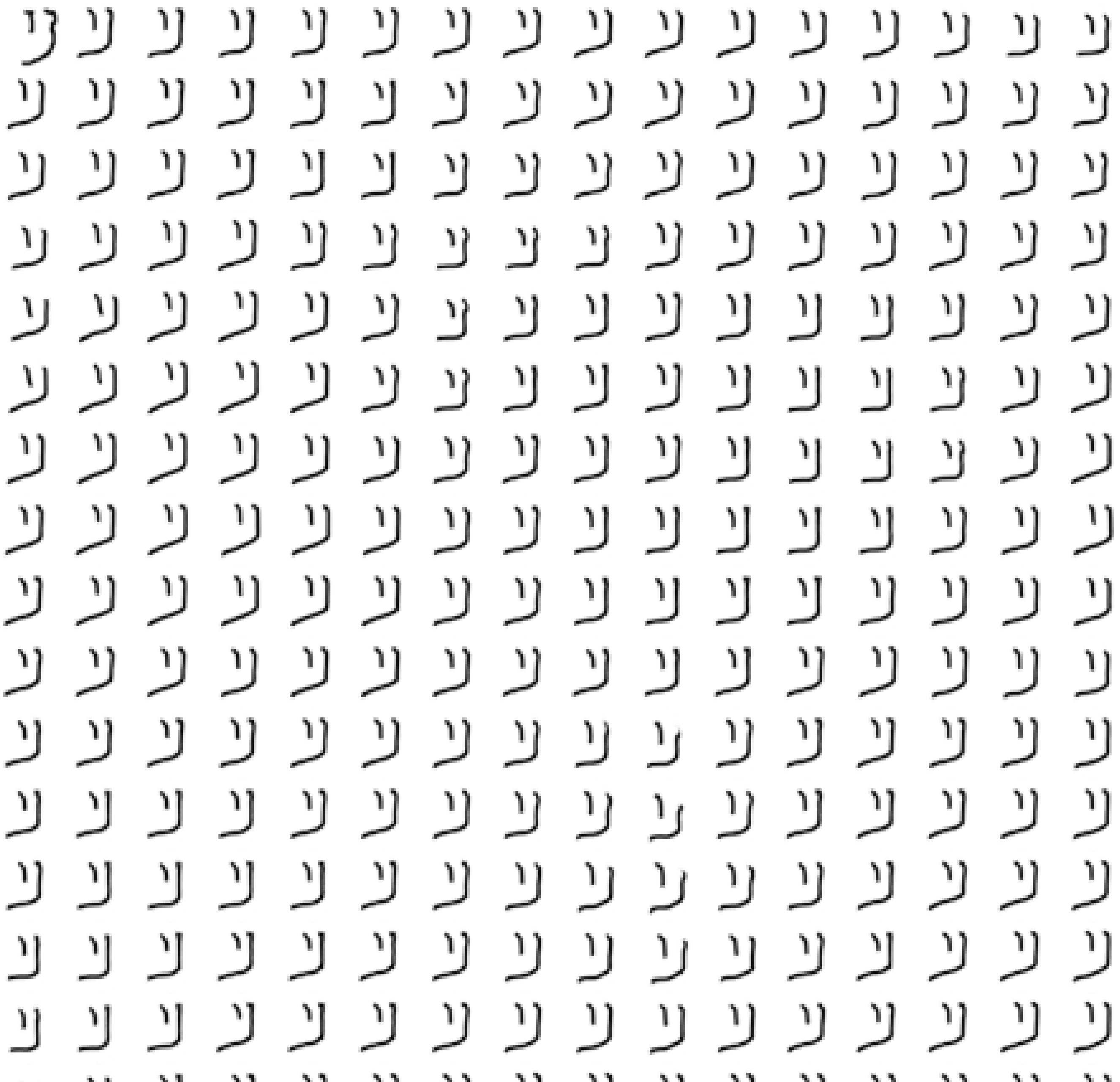

Figure 3: The letter ”y” generated.

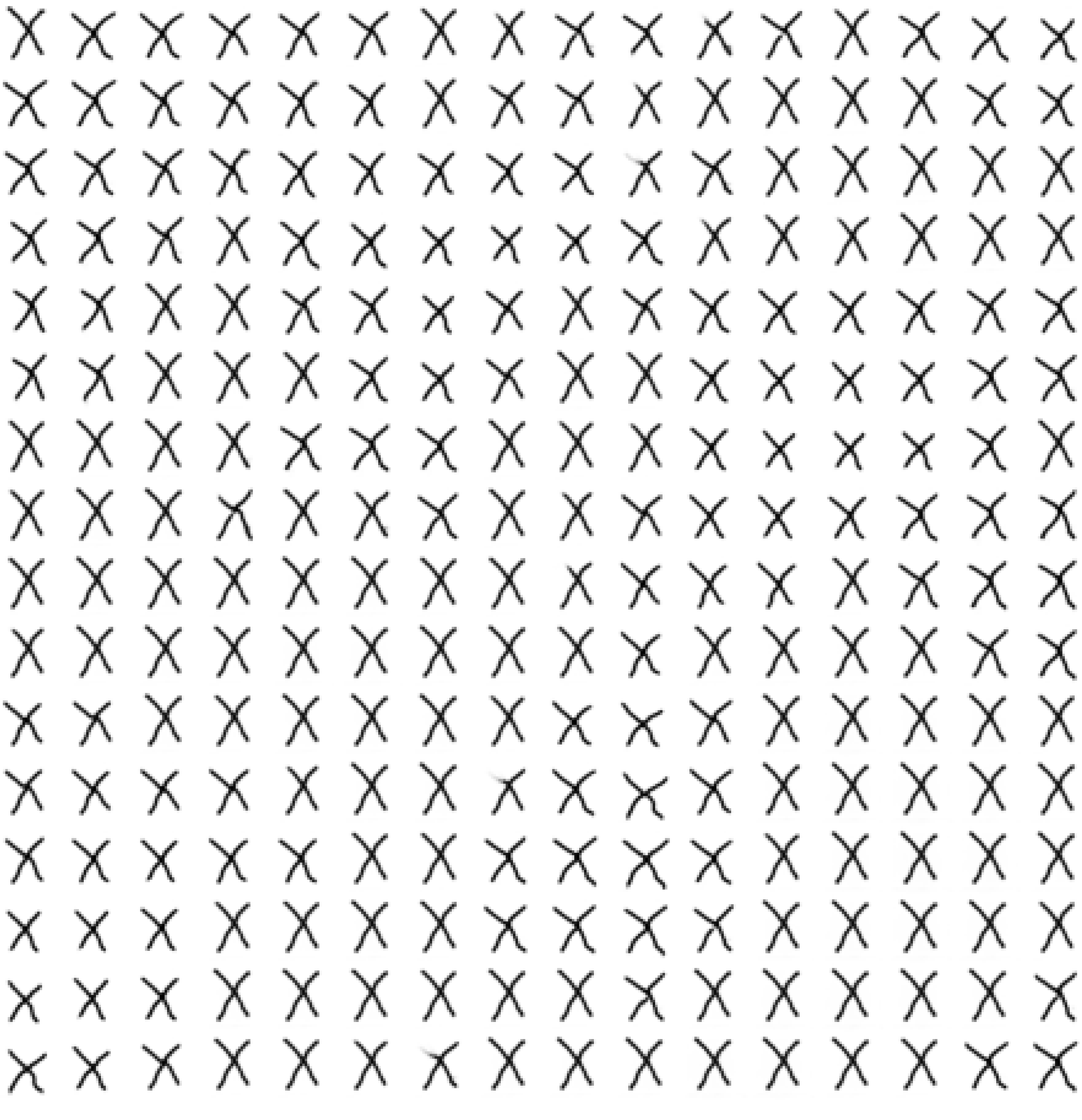

Figure 4: The letter ”x” generated.

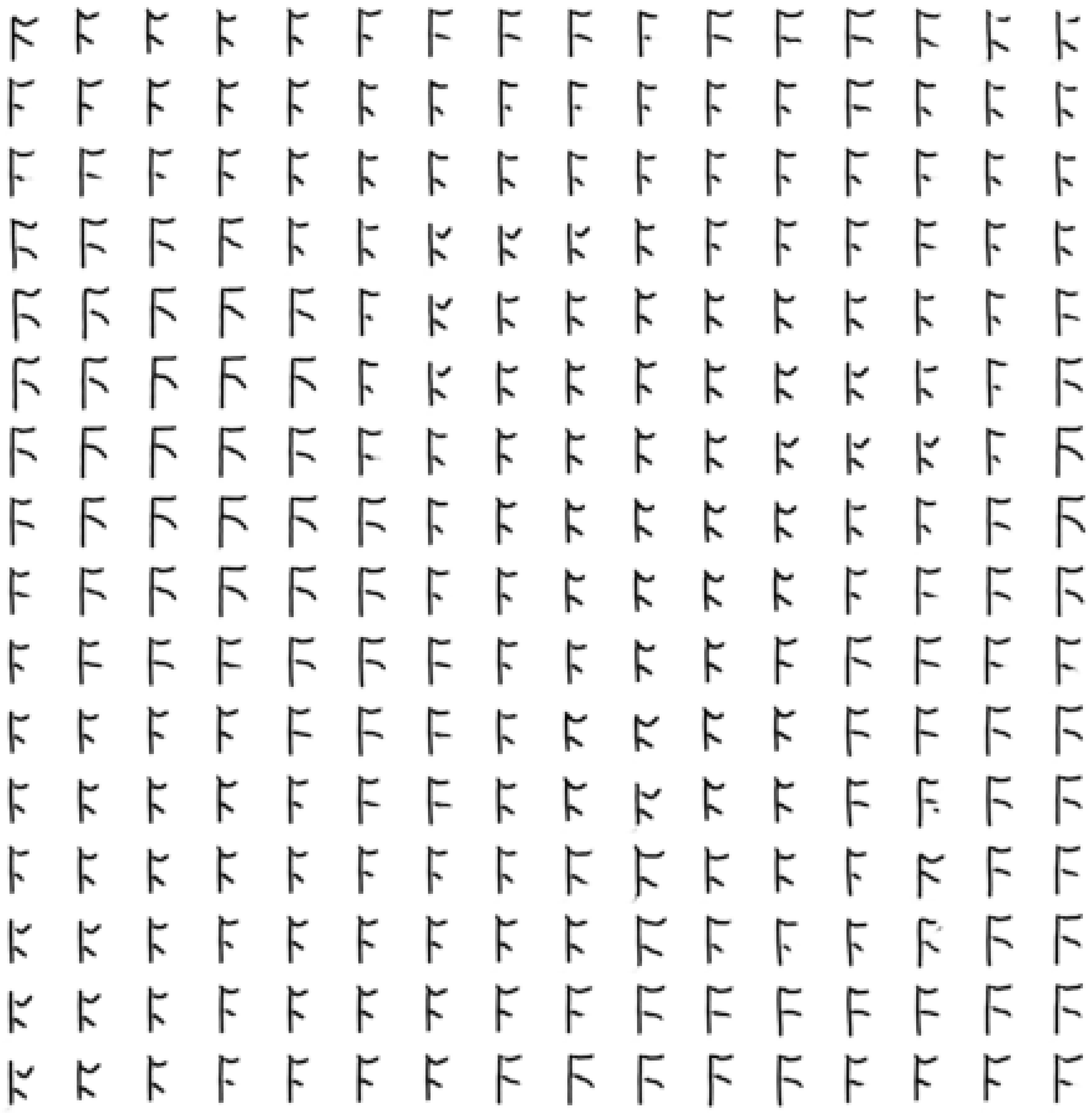

Figure 5: The letter "F" generated.