# The Efficacy of Reversible Flow Models for Improved Biometric Verification Systems

**Christopher Waites (006358440)**
Department of Computer Science
Stanford University
Stanford, CA 94305
`waites@stanford.edu`

`http://www.github.com/ChrisWaites/go-with-the-flow/`

## 1 Introduction

User verification is considered a foundational component to many digital systems deployed in existence today. As such, designing such systems while holding them to high standards of security and usability is of the utmost importance. It goes without saying that optimizing for security will always be a valuable attribute of such a system, and continued improvements in this regard will be a never-ending study. Although, in recent years, the latter point concerning usability has seen significantly more innovation than historically. Much of this is motivated by a guiding principle that verification, at it's core, is the task of convincing a machine you are the identity you claim to be. In this respect, biometric authentication has garnered more attention given its reasonable tradeoff between security and usability, as collecting a biometric sample is relatively seamless as compared to, say, recalling a cryptographic hash. Many would believe that this is due to the rise of machine learning, that is our improved ability to work effectively with arbitrary, human-centric distributions with little analytical or theoretical foundations. By using biometric readings as an input to a user verification system, one could hope to develop systems which are able to more seamlessly and unobtrusively infer the identity of their user and selectively allow access to certain functionality.

On a high level, the task of user verification is simply a binary classification task in which the system in question is tasked with classifying whether or not a given biometric reading corresponds to a claimed identity. A natural approach to this is via density estimation, in which two generative models are trained: one to approximate the distribution of biometric readings for the population, and one to approximate the distribution of biometric readings for the identity. Given these approximated distributions, assuming they allow for efficient querying of the density, one is able to go forward and calculate the relative likelihood of the observed data under either model and make an assertion as to whether the data is more likely to have come from the population or the claimed identity.

The Gaussian Mixture Model-Universal Background Model (GMM-UBM) framework is the current predominating methodology backing many biometric verification systems today, originally proposed for speaker verification. GMM-UBM is comprised of a single, large GMM trained to approximate the population distribution of text-independent speech features and is employed as the expected alternative speaker model during the task of verification. That is, when conducting the log-likelihood ratio test, it is used as a prior for general human speech to compare against the likelihood of the data under the user model.

Although in recent years, well after the development of many user verification systems deplyed today, there have been significant developments in improved techniques for generative modeling which allow for efficient density queries, in particular, reversible flow models. Given their additional expressivity and ability to converge on complex high-dimensional distributions, there should be reason to believe that improved performance of user verification systems could come from replacing GMM-UBM with reversible flow models. In this project we investigate this hypothesis, exploring the efficacy of reversible flow models as a replacement for GMM-UBM in the task of biometric user verification.

## 2 Related Work

*Speaker Verification Using Adapted Gaussian Mixture Models* [10] is the primary source of work from which our improvements stem. In this work, Reynolds et al. introduce the original model for GMM-UBM, built around the log-likelihood ratio test for verification, using rather simplistic but effective GMMs for likelihood functions. They propose the use of a universal background model for alternative speaker representation and a form of Bayesian adaptation to derive speaker models from the UBM, improving the quality of the resulting user models given that data is usually sparse during user enrollment. They were able to demonstrate promising empirical performance metrics on representative performance benchmarks and system behavior experiments, namely on the NIST SRE corpora.

*A Study on Universal Background Model Training in Speaker Verification* [5] acts as a good suvery of the current state of GMMs as applied to biometric verification, in particular for speaker verification. This has thus far provided useful guidance concerning the contexts and standards to which these models are generally trained, the nuances and conventions involved in working with biometric data, and proper metrics for evaluating such models. Although, many of the techniques described are rather traditional in comparison to the deep learning methods which are often preferred today.

With respect to work which motivate our proposed methods on an algorithmic level, *NICE: Non-linear Independent Components Estimation* [1], *MADE: Masked Autoencoder for Distribution Estimation* [3], *Density Estimation Using Real NVP* [2], *Glow: Generative Flow with Invertible 1×1 Convolutions* [6], and *FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models* [4] all propose algorithms which adhere to the interface of the task in question, and have the potential to yield benefits over Gaussian mixture models due to increased expressivity. Each, using their own nuanced strategy, attain the high-level objective of transforming some simplistic, prior distribution via a series of reversible operations and allow for querying the density of a given sample, necessary to perform the log-likelihood ratio test when verifying a user.

## 3 Problem Statement

### 3.1 User Verification Task

The task of identity verification for a given biometric sample $X = \{x_1, \ldots, x_T\}$ and claimed identity $U$ can be stated as a simple hypothesis test between $H_0$: $X$ is from the hypothesized user $U$, and $H_1$: $X$ is *not* from the hypothesized user $U$. The optimum test [9], a likelihood ratio test, is used to decide as to whether or not to accept the null hypothesis, characterized by the decision rule stated below.

$$\frac{p(X|H_0)}{p(X|H_1)} = \begin{cases} \geq \mathcal{T} & \text{accept } H_0 \\ < \mathcal{T} & \text{reject } H_0 \end{cases}$$

Observe that this rule depends upon the probability of the observed sample under an assumption of hypothesis, and depends upon a decision parameter $\mathcal{T}$, often selected empirically to achieve some desired false positive rate depending on the context of the system.

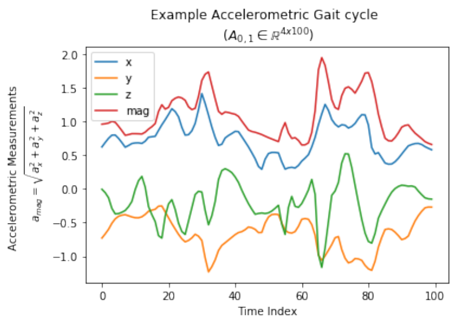
Thus far we have assumed access to the true data distribution, which is generally not available in the real world. So, to make this hypothesis test actionable, we assume that we can model parametrically the distribution of realistic biometric readings from the user and the general population. That is, assume we are able train two models, parameterized by $\theta_u$ and $\theta_p$ respectively, characterizing the probability distribution over biometric readings for the given user $U$ and the general population. Then, one could feasibly perform the log-likelihood ratio test for a sequence of observed feature vectors $X$, computed as the following.

$$\Lambda(X) = \log p_{\theta_u}(X) - \log p_{\theta_u}(X)$$

Then, one could simply accept or reject the null hypothesis depending upon whether $\Lambda(X)$ exceeds some set threshold $\mathcal{T}$.

### 3.2 Dataset

In our context, references to biometric readings specifically correspond to Gait cycles from the GaitNet dataset, a novel dataset consisting of accelerometer data pulled from the handheld devices of 1,000 users across 117 countries and 1297 cities, acting as input into our system. To be concrete, a Gait cycle roughly corresponds to accelerometric readings (namely, the components of the acceleration vector of the phone, by convention also including the magnitude) over roughly two human steps, which would yield an individual example of the form $x \in \mathcal{R}^{4 \times 100}$. Each user contributed roughly 100 training trajectories and 10 test trajectories, yielding a test set $X_{train} \in \mathcal{R}^{999904 \times 4 \times 100}$ and $X_{test} \in \mathcal{R}^{99999 \times 4 \times 100}$. Class labels for each cycle are also provided, each taking on some value in the range $\{0, 1, \ldots, 999\}$ denoting the ID of the user. A visualization of an arbitrarily selected trajectory is included for convenience below.



Despite the dataset having a clear temporal structure, we found empirically that capturing some notion of underlying, recurrent state properly (using RNNs, for example) was relatively impractical given the noise and variance exhibited in the data. Throughout our work thus far, our models have opted to rather perform a convolution over the temporal dimension of the data, and we have found that this has yielded much more promising results. Although, given additional time and data, modeling the data with models which use a recurrent structure may be an interesting direction to pursue in the future.

Finally to provide additional context, given the recent and novel nature of the dataset in question, the original curators have been able to verify with certainty that an application of reversible generative models to perform user verification in the proposed manner would be a novel application which has not been performed before. It is also believed that, to a reasonable degree of certainty, an application of reversible generative models to the task of user verification using the log-likelihood test in the proposed manner in general is, if not novel, an emerging field of study.

## 4 Approach

### 4.1 Feature Extraction

Rather than performing statistical analysis directly on raw data as given, oftentimes better performance can be achieved by performing analysis on more efficient representations of the data which are thought to convey more essential information concerning the sample. For example in this context, it could potentially be more practical for our generative model to learn over vectors which include some notion of frequency, extremity, etc. rather than having to learn such qualities indirectly given low-level information concerning qualities such as the X-component of the acceleration vector at time $t$ being precisely 0.1452 $m/s^2$.

A common manner to approach feature extraction, as was done in our context, was to construct a classification network with one output for each unique user ID. We simply trained a model to perform user classification given the dataset $(X_{train}, Y_{train})$ of raw biometric readings and their corresponding user IDs as labels, optimizing for categorical cross entropy loss. As will be shown later, performing this classification in a manner which exploits the temporal nature of the data using a recurrent model architecture deemed impractical. Thus, we opted for a network which performed classification via a temporal convolution, that is, a convolution over the temporal dimension of the data treating the components of the acceleration vector as channels. Full model details can be found in the appendix.

```
----------------------------------------------------------------------------------------------------
Layer (type) Output Shape Param # Connected to
====================================================================================================
main_input (InputLayer) (None, 1, 4, 100) 0
6
----------------------------------------------------------------------------------------------------
conv2d_9 (Conv2D) (None, 64, 4, 96) 384 main_input[0][0]
...
batch_normalization_16 (BatchNo (None, 512) 2048 dense_5[0][0]
----------------------------------------------------------------------------------------------------
dense_6 (Dense) (None, 1000) 513000 batch_normalization_16[0][0]
====================================================================================================
Total params: 2,150,824
Trainable params: 2,148,504
Non-trainable params: 2,320
```

Given this trained model, the hypothesis is that for the model to perform well at its respective task, it should be picking up on some more efficient representation of the data in one of the intermediary layers of the network. Under this pretense, once the model is trained, we remove the last $k$ layers (in our case, the last two layers), and pass future examples through this augmented network to act as a feature extractor.

$$\begin{bmatrix} \dots \\ \dots \\ \dots \end{bmatrix} \in \mathbb{R}^{N\times 4\times 100} \xrightarrow{FeatureExtraction} \begin{bmatrix} \dots \\ \dots \\ \dots \end{bmatrix} \in \mathbb{R}^{N\times 512}$$

An unnecessary but useful byproduct of this approach is that it also makes the proposed system invariant to the precise nature of the biometric readings. That is, the modeling performed in subsequent steps only makes the assumption that it receives biometric reading feature vectors, rather than relying on some key component concerning Gait cycles specifically. Hence, the proposed approach is equally applicable to fingerprint data assuming some effective feature extraction network can be constructed, likely following the exact same principle of training a network to perform user classification and intercepting useful intermediate representations.

## 4.2 GMM-UBM

In the GMM-UBM framework, the user model is adapted using the parameters of the population model using a form of Bayesian adaptation. Unlike the straightforward approach of performing maximum likelihood training of a model for the user independent of the population model, the basic idea in adaptation is to derive the user's model by updating the well-trained parameters of the population model to incorporate user data. This yields a tighter coupling between the user's model and the population model which produces better performance than decoupled models due to user data sparsity.

To make this process more concrete, similar to expectation maximization, adaption is a two step estimation process. The first step is identical to the expectation step of expectation maximization, where estimates of the sufficient statistics of the user's training data are computed for each mixture in the Gaussian mixture model. Although, unlike the second step of the EM algorithm, the new sufficient statistic estimates are then combined with the old sufficient statistics from the population model parameters using a context-dependent weighting coefficient.

Algorithmically, for mixture $i$ in the GMM, we compute the following:

$$Pr(i|\boldsymbol{x}_i) = \frac{w_i p_i(\boldsymbol{x}_t)}{\sum_{j=1}^{M} w_j p_j(\boldsymbol{x_t})}$$

We then use $Pr(i|\boldsymbol{x}_t)$ and $\boldsymbol{x}_t$ to compute the sufficient statistics for the weight, mean, and variance parameters:

$$n_i = \sum_{t=1}^{T} Pr(i|\boldsymbol{x}_t)$$

$$E_i(\boldsymbol{x}_i) = \frac{1}{n_i}\sum_{t=1}^{T} Pr(i|\boldsymbol{x}_t)\boldsymbol{x}_t$$

$$E_i(\boldsymbol{x}_i^2) = \frac{1}{n_i}\sum_{t=1}^{T} Pr(i|\boldsymbol{x}_t)\boldsymbol{x}_t^2$$

Then, new sufficient statistics from the user data are used to update the old GMM sufficient statistics for mixture $i$ to create the adapted parameters via the following equations:

$$\hat{w}_i = [\alpha_i^w n_i/T + (1-\alpha_i^w)w_i]\,\gamma$$
$$\hat{\boldsymbol{\mu}}_i = \alpha_i^m E_i(\boldsymbol{x}) + (1-\alpha_i^m)\boldsymbol{\mu}_i$$
$$\hat{\boldsymbol{\sigma}}^{\boldsymbol{2}}{}_i = \alpha_i^v E_i(\boldsymbol{x}^2) + (1+\alpha_i^v)(\boldsymbol{\sigma}_i^2 + \boldsymbol{\mu}_i^2) - \hat{\boldsymbol{\mu}}_i^2$$

## 4.3 Normalizing Flow Models

Normalizing flow models are a class of generative models which allow for tractable density queries. They achieve this by applying a series of invertible transformations to some known prior distribution with an analytical probability density function. Given that the transformations are defined to be invertable, one can calculate densities of points in the resulting, transformed distribution via the change of variables formula, detailed below.

$$p_X(x) = p_Z(f^{-1}(x))\left|det\left(\frac{\partial f^{-1}(x)}{\partial x}\right)\right|$$

Given that density querying is feasible, one is able to explicitly optimize the model for the log-likelihood of the training data. Hence, given biometric reading data from an enrolling user and the general population, we can go forward and begin to approximate the biometric reading distribution over each, $p_{\theta_u}(x)$ and $p_{\theta_p}(x)$.

As stated previously, biometric verification systems generally opt to incorporate some notion of adaption when training to address the fact that, in practical contexts, one typically has receives very few user examples during enrollment. Recall that this is done for UBM-GMM via a simple augmentation on the expectation maximization algorithm to optimize for a weighted sum of the parameters inferred by the user data and the population model parameters.

Although, in their canonical formulation, our models of interest are not trained according to expectation maximization, but are rather trained to explicitly minimize the negative log-likelihood of the data it's given. To incorporate some notion of adaptation into our framework, the most natural augmentation available is to change the loss function to somehow include the aforementioned $\alpha$ weighting. That is, we can define the loss function of our model to be the following, where $U$ is the dataset of the user's biometric data, $P$ is a dataset of general population biometric readings, and $\alpha$ weights the relative importance of optimizing the likelihood of the user's data versus the general population data.

$$\mathcal{L}(X;\theta_u) = -\textstyle\sum_{x_i\sim p_U(x)} \log p_{\theta_u}(x_i) - \alpha \sum_{x_j\sim p_P(x)} \log p_{\theta_u}(x_j)$$

Given this new loss function which incorporates some notion of adaptation, we can go forward to train the user generative model parameterized by $\theta_u$ by minimizing this loss function, and train the general population generative model parameterized by $\theta_p$ by minimizing regular negative log-likelihood over $U$ alone. Given these two models, and the fact that density queries are efficient by design, we can use them to approach the task of user verification using the log-likelihood ratio test.

Intuition for why implicitly the forward KL-divergence is used as opposed to the reverse KL-divergence follows as such. Intuitively, it is preferable in this context to only spread the density of $p_{\theta_u}(x)$ over regions where $p_{\theta_p}(x)$ is nonzero. That is, given that the user is a member of the population itself and should in theory lie within this distribution, when optimizing $p_{\theta_u}(x)$, we should not inadvertently make regions of unrealistic biometric readings suddenly be reasonable for the user to take on.

The given codebase expands upon the details of training in additional detail, but roughly the models were trained with a learning rate of $1e-4$ with a weight decay of $1e-5$ and optimized via Adam for the aforementioned loss function, either explicitly or implicitly through extending the dataset it was trained on to achieve the desired weighting.

## 5 Results

### 5.1 Experimental Procedure

The feature extraction network was retrieved by training the aforementioned classification model to perform user ID classification on $X_{train}$ from the GaitNet dataset. During training, we used a batch size of 1024, a cosine annealing learning rate scheduler with periodic restarts [12], and Adam to optimize for categorical cross entropy. Then, using this network, various performance measures were collected on the holdout set $X_{test}$ and displayed in 1. The feature extraction network was then attained by removing the last two layers of the network.

Then, to evaluate the performance of different generative modeling methods, we simulated the enrollment process across users. That is, we repeated and averaged the following over each user $i \in \{0, \ldots, 999\}$, thereby accounting for individual variance. First, a generative model was trained on the complement of the user data $X_{train} \setminus X^i_{train}$ to yield the population model $p_{\theta_p}(x)$. Then the generative model, depending upon its type (GMM-UBM vs. Flow), was adapted to the user model by having access to $X^i_{train}$ to yield $p_{\theta_u}(x)$. Finally, we used both models to calculate the average log-likelihood they assign to holdout vectors across three data sources, namely $X_{test} \setminus X^i_{test}$ (corresponding to holdout non-user data), $X^i_{train}$ (corresponding to holdout user data), and $X_{sinusoidal}$ (corresponding to random sinusoidal signals). Results from this process are given in 2.

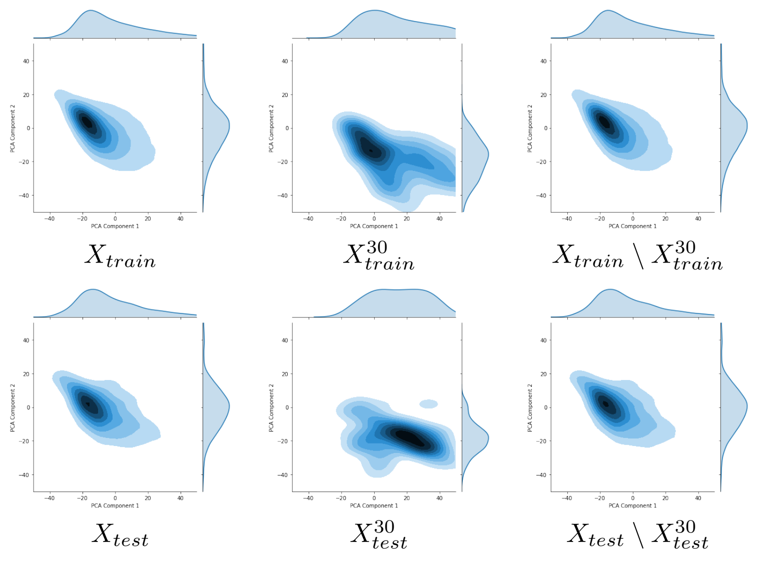


Figure 1: Example distributions which would be approximated during the simulated enrollment process. This includes the population distributions $X_{train}$ and $X_{test}$, the distributions for a randomly selected user $X^i_{train}$ and $X^i_{test}$, and the population distributions with this user removed $X_{train} \setminus X^i_{train}$ and $X_{train} \setminus X^i_{test}$.

### 5.2 Results

As seen in 1, the temporal CNN architecture seems to outperform alternative methods. Although, it's generally helpful to interpret such results with respect to some appropriate baseline. In particular, recalling that the task at hand was a 1000-class classification problem, a random selection criterion would naturally only yield 0.01% accuracy in expectation. Given this, there's reason to believe that 57.92% is at the very least a nontrivial result, supported by the significant improvements gained from choices in hyperparameters and architecture. In addition, looking at the difficulty of the task from a higher level, recall that we are performing user classification only given trajectories corresponding to about one to two steps of human motion. Intuitively this is quite difficult, and in a practical context one would likely have access to a longer sequence from which to make a prediction which should improve performance.

Table 1: Performance of classification network (feature extractor) on 1000-class classification task, trained on $X_{train}, y_{train}$ and evaluated on $X_{test}, y_{test}$.

| Model | Accuracy | Precision | Recall | $F_1$ Score |
|---|---|---|---|---|
| Feed-Forward MLP | 0.2837 | 0.2065 | 0.2146 | 0.2105 |
| RNN | 0.4331 | 0.3131 | 0.3521 | 0.3315 |
| Temporal CNN | 0.5792 | 0.5321 | 0.5635 | 0.5474 |

To justify the motivation for the metric detailed in 2, it essentially addresses the question: "If I use generative modeling technique X to train a population and user model, what likelihood should I expect to receive when I give it population data, the user's data, and noise?" This is of interest because it acts as a succinct summarization of the model's expected behavior, and the resulting values are capable of answering several questions of interest. For example, is the likelihood assigned to user data by the user model expected to be significantly higher than the likelihood assigned to user data by the population model? This would be preferable, as this would correspond to a model which is able to resist false negatives, i.e. failing to verify a user given user data. Similarly, if the likelihood assigned to population data by the user model is approximately the same or lower than the likelihood assigned to population data by the population model, then the model would be expected to resist some notion of false positives, i.e. inappropriately verifying a member from the population as the user. Finally, one would also be able to identify the propensity of the model to discriminate between realistic and fake readings altogether if, in expectation, it were to assign random signals very low likelihood.

One will find that each of these properties can be observed in 2 to varying degrees, depending upon model under consideration. In particular, reversible flow models seem to perform preferably over GMM acting as our baseline. In particular, the RealNVP model we were able to train seems to assign the lowest likelihood to pronounce the likelihood of user data in the user model the most and assign the lowest likelihoods to random sinusoidal waves. This might follow one's intuition as one could claim that, in some sense, RealNVP strikes a quite reasonable tradeoff between ease of training and expressivity in comparison to some of the other models considered, and the distribution of application is not overtly complicated in comparison to, say, some image datasets.

Table 2: Log-likelihood assigned to holdout data from varying data sources by model, averaged over all users taken as enrollee.

| Model | Type | $x \sim$ Population | $x \sim$ User | $x \sim$ Sinusoidal |
|---|---|---|---|---|
| GMM | Population | 1568.9 | 1701.2 | -713.1 |
| GMM | User Adapted | 1568.7 | 1717.3 | -711.4 |
| MAF | Population | 1650.1 | 1758.9 | -701.3 |
| MAF | User Adapted | 1658.4 | 1788.9 | -695.2 |
| Glow | Population | 1788.4 | 1832.1 | -675.8 |
| Glow | User Adapted | 1786.3 | 1856.3 | -675.1 |
| RealNVP | Population | 1723.5 | 1801.6 | -651.5 |
| RealNVP | User Adapted | 1721.6 | 1862.1 | -653.3 |

## 5.3 Analysis

To perform a qualitative evaluation of our models, we opted to draw a sufficiently large sample from each, map each 512-dimensional vector to two dimensions via PCA (fit to $X_{train}$), plotted the resulting distributions.

Metrics for PCA when fit to $X_{train}$ are shown in 3.

After interpreting the metrics collected from PCA, this offers a plausible explanation for why the quantitative metrics and qualitative metrics seem to infer contradictory conclusions. That is, by many

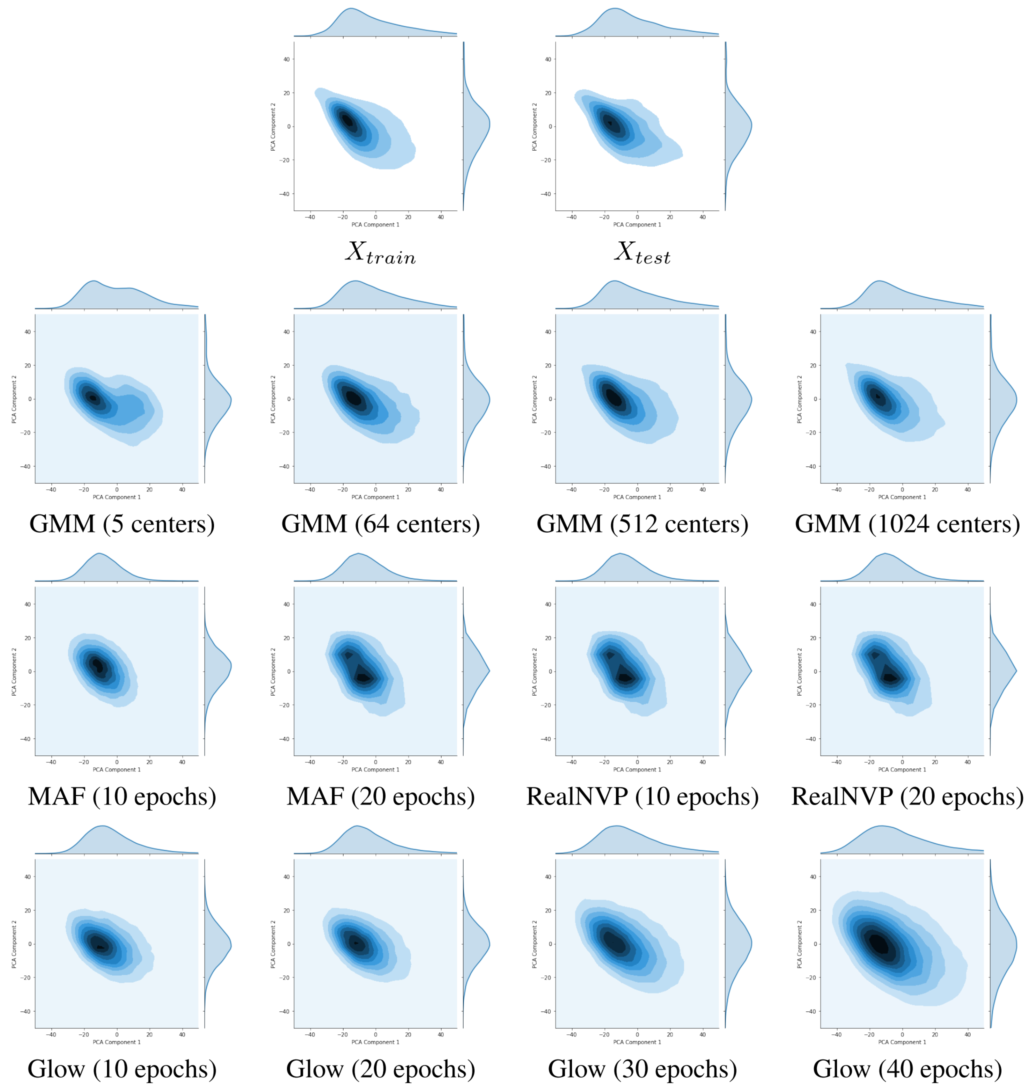


Figure 2: Approximated distributions for varying model type, plotted on axes derived via PCA on $X_{train}$.

Table 3: Various PCA metrics by component when fit to $X_{train}$ for visualization.

| Metric | PCA Component 1 | PCA Component 2 |
|---|---:|---:|
| Singular Values | 7076.8780 | 5632.2750 |
| Explained Variance | 500.8270 | 317.2283 |
| Explained Variance Ratio | 0.0883 | 0.0559 |

standards, the principal components yielded by PCA seem to explain rather little variance in the data. That is, the offered plots can likely guide some very shallow deductions about the quality of the approximated distributions, although, there seems to be nontrivial variance outside what can be expressed in two dimensions.

## 6 Conclusion

In this work, we were able to explore the efficacy of reversible flow models in improving existing biometric verification frameworks, in particular GMM-UBM, as applied to the GaitNet dataset. We were able to improve upon the techniques proposed by GMM-UBM via the use of more expressive reversible flow models, and validated our results quantitatively and qualitatively via out-of-sample log-likelihood and distribution visualization.

For future work, it would be interesting to investigate the use of conditional generative models to perform the task of user recognition. That is, the ability to not only identify whether or not a given biometric reading corresponds to a set identity, but to be able to select the most likely identity for a given reading out of a population given labels.

Additionally, our feature recognition system was done via temporal convolutions over the Gait trajectories. Assuming there is some more efficient, underlying recurrent representation of the data, it would be interesting to replace this convolutional feature extraction step with a recurrent one even though our preliminary evaluations inferred that the features extracted via convolution were of reasonable enough quality to not require this.

## References

[1] Laurent Dinh, David Krueger, and Yoshua Bengio. *NICE: Non-linear Independent Components Estimation*. 2014. arXiv: `1410.8516 [cs.LG]`.

[2] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. “Density estimation using Real NVP”. In: *CoRR* abs/1605.08803 (2016). arXiv: `1605.08803`. URL: `http://arxiv.org/abs/1605.08803`.

[3] Mathieu Germain et al. “MADE: Masked Autoencoder for Distribution Estimation”. In: *CoRR* abs/1502.03509 (2015). arXiv: `1502.03509`. URL: `http://arxiv.org/abs/1502.03509`.

[4] Will Grathwohl et al. “FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models”. In: *CoRR* abs/1810.01367 (2018). arXiv: `1810.01367`. URL: `http://arxiv.org/abs/1810.01367`.

[5] Taufiq Hasan and John H. L. Hansen. “A Study on Universal Background Model Training in Speaker Verification”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 19.7 (Sept. 2011), pp. 1890–1899. DOI: `10.1109/TASL.2010.2102753`.

[6] Diederik P. Kingma and Prafulla Dhariwal. *Glow: Generative Flow with Invertible 1x1 Convolutions*. 2018. arXiv: `1807.03039 [stat.ML]`.

[7] Ilya Kostrikov. *Pytorch Flows*. 2018. URL: `https://github.com/ikostrikov/pytorch-flows`.

[8] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. “WaveGlow: A Flow-based Generative Network for Speech Synthesis”. In: *CoRR* abs/1811.00002 (2018). arXiv: `1811.00002`. URL: `http://arxiv.org/abs/1811.00002`.

[9] D. A. Reynolds and R. C. Rose. “Robust text-independent speaker identification using Gaussian mixture speaker models”. In: *IEEE Transactions on Speech and Audio Processing* 3.1 (Jan. 1995), pp. 72–83. DOI: `10.1109/89.365379`.

[10] Douglas A. Reynolds, Thomas F. Quatieri, and Robert B. Dunn. “Speaker Verification Using Adapted Gaussian Mixture Models”. In: *Digit. Signal Process.* 10.1 (Jan. 2000), pp. 19–41. ISSN: 1051-2004. DOI: `10.1006/dspr.1999.0361`. URL: `http://dx.doi.org/10.1006/dspr.1999.0361`.

[11] *seaborn.jointplot*. 2018. URL: `https://seaborn.pydata.org/generated/seaborn.jointplot.html`.

[12] *Setting the Learning Rate of Your Neural Network*. Mar. 2018. URL: `https://www.jeremyjordan.me/nn-learning-rate/`.

## 7 Appendix

### 7.1 Model Architecture

Classification and Feature Extraction Network

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
main_input (InputLayer)         (None, 1, 4, 100)    0
__________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 64, 4, 96)    384         main_input[0][0]
__________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 64, 4, 96)    384         conv2d_9[0][0]
__________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 64, 1, 92)    81984       batch_normalization_9[0][0]
__________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 64, 1, 92)    368         conv2d_10[0][0]
__________________________________________________________________________________________________
conv2d_11 (Conv2D)              (None, 64, 1, 92)    16448       batch_normalization_10[0][0]
__________________________________________________________________________________________________
batch_normalization_11 (BatchNo (None, 64, 1, 92)    368         conv2d_11[0][0]
__________________________________________________________________________________________________
add_6 (Add)                     (None, 64, 1, 92)    0           batch_normalization_10[0][0]
                                                                 batch_normalization_11[0][0]
__________________________________________________________________________________________________
conv2d_12 (Conv2D)              (None, 64, 1, 92)    32832       add_6[0][0]
__________________________________________________________________________________________________
batch_normalization_12 (BatchNo (None, 64, 1, 92)    368         conv2d_12[0][0]
__________________________________________________________________________________________________
add_7 (Add)                     (None, 64, 1, 92)    0           add_6[0][0]
                                                                 batch_normalization_12[0][0]
__________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, 64, 1, 92)    32832       add_7[0][0]
__________________________________________________________________________________________________
batch_normalization_13 (BatchNo (None, 64, 1, 92)    368         conv2d_13[0][0]
__________________________________________________________________________________________________
add_8 (Add)                     (None, 64, 1, 92)    0           add_7[0][0]
                                                                 batch_normalization_13[0][0]
__________________________________________________________________________________________________
conv2d_14 (Conv2D)              (None, 64, 1, 92)    32832       add_8[0][0]
__________________________________________________________________________________________________
batch_normalization_14 (BatchNo (None, 64, 1, 92)    368         conv2d_14[0][0]
__________________________________________________________________________________________________
add_9 (Add)                     (None, 64, 1, 92)    0           add_8[0][0]
                                                                 batch_normalization_14[0][0]
__________________________________________________________________________________________________
conv2d_15 (Conv2D)              (None, 64, 1, 92)    32832       add_9[0][0]
__________________________________________________________________________________________________
batch_normalization_15 (BatchNo (None, 64, 1, 92)    368         conv2d_15[0][0]
__________________________________________________________________________________________________
add_10 (Add)                    (None, 64, 1, 92)    0           add_9[0][0]
                                                                 batch_normalization_15[0][0]
__________________________________________________________________________________________________
lambda_3 (Lambda)               (None, 1, 64, 92)    0           add_10[0][0]
__________________________________________________________________________________________________
average_pooling2d_2 (AveragePoo (None, 1, 4, 33)     0           main_input[0][0]
__________________________________________________________________________________________________
conv2d_16 (Conv2D)              (None, 1, 64, 32)    23584       lambda_3[0][0]
__________________________________________________________________________________________________
locally_connected2d_2 (LocallyC (None, 16, 1, 8)     2176        average_pooling2d_2[0][0]
__________________________________________________________________________________________________
flatten_3 (Flatten)             (None, 2048)         0           conv2d_16[0][0]
__________________________________________________________________________________________________
flatten_4 (Flatten)             (None, 128)          0           locally_connected2d_2[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 512)          1049088     flatten_3[0][0]
__________________________________________________________________________________________________
lambda_4 (Lambda)               (None, 128)          0           flatten_4[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 640)          0           dense_4[0][0]
                                                                 lambda_4[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 512)          328192      concatenate_2[0][0]
__________________________________________________________________________________________________
batch_normalization_16 (BatchNo (None, 512)          2048        dense_5[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 1000)         513000      batch_normalization_16[0][0]
==================================================================================================
Total params: 2,150,824
Trainable params: 2,148,504
Non-trainable params: 2,320
```