# DisUnknown: Distilling Unknown Factors for Disentanglement Learning

Sitao Xiang[1,2], Yuming Gu[1,2], Pengda Xiang[1,2], Menglei Chai[3], Hao Li[2], Yajie Zhao[2], Mingming He[2]
[1]University of Southern California, [2]USC Institute for Creative Technologies, [3]Snap Inc.
sitaoxia@usc.edu, {ygu, pxiang}@ict.usc.edu, mchai@snap.com, hao@hao-li.com,
{zhao, he}@ict.usc.edu

## Contents

# 1. Method Details

## 1.1. Derivation of the Negative Log Unlikelihood

Here we show how the form of the adversarial loss of the classifiers are chosen.

First, we consider the assumed equilibrium of the adversarial training. In Stage I, the goal is that the encoder's output should not contain any information about the labeled factors. So, each classifier $C_i$ can at best make a guess, and since there is no way to distinguish between inputs from different classes, it should give the same output class distribution for every sample.

Assume that, as commonly done, the output distribution of the classifier is computed by taking the softmax of a vector $t = (t_{(1)}, t_{(2)}, \ldots, t_{(m)})$, where $m$ is the number of classes for the factor of concern. Let $\mathrm{Sm}$ denote softmax, we have:

$$\begin{aligned}
&\frac{\partial}{\partial t_{(i)}}\mathsf{NLL}(\mathrm{Sm}(t), k) \qquad (1)\\
=&\frac{\partial}{\partial t_{(i)}} - \ln \frac{e^{t_{(k)}}}{\sum_j e^{t_{(j)}}}\\
=&\frac{\partial}{\partial t_{(i)}}(-t_{(k)} + \ln \sum_j e^{t_{(j)}})\\
=&-\delta_{ik} + \frac{e^{t_{(i)}}}{\sum_j e^{t_{(j)}}}\\
=&-\delta_{ik} + \mathrm{Sm}(t, i),
\end{aligned}$$

where $\delta_{ik} = 1$ when $i = k$ and $\delta_{ik} = 0$ otherwise. At equilibrium, the expectation of gradient over the whole dataset should be zero. Let $q = (q_{(1)}, q_{(2)}, \ldots, q_{(m)})$ be the class frequency in the dataset. Then we must have

$$\begin{aligned}
&\frac{\partial}{\partial t_{(i)}}\sum_k q_{(k)}\mathsf{NLL}(\mathrm{Sm}(t), k) \qquad (2)\\
=&\sum_k q_{(k)}(-\delta_{ik} + \mathrm{Sm}(t, i))\\
=&\mathrm{Sm}(t, i) - q_{(i)}\\
=&0.
\end{aligned}$$

That is, $\mathrm{Sm}(t, i) = q_i$. So, at the assumed equilibrium, the classifier should give the class distribution in the dataset as the output for any input.

If the adversarial objective is to maximize the NLL loss of the classifier, then naturally at this assumed equilibrium the expected gradient of the adversarial loss function is also zero. But, consider the second-order derivatives:

$$\begin{aligned}
&\frac{\partial^2}{\partial t_{(i)}^2}\mathsf{NLL}(\mathrm{Sm}(t), k) \qquad (3)\\
=&\frac{\partial}{\partial t_{(i)}}(-\delta_{ik} + \mathrm{Sm}(t, i))\\
=&\mathrm{Sm}(t, i)(1 - \mathrm{Sm}(t, i))\\
>&0,
\end{aligned}$$

the adversarial loss function has a local minimum with respect to $t$ at the assumed equilibrium, while the objective is to maximize this function. So in the proximity of the assumed equilibrium, the adversarial objective actually pushes the networks away from the equilibrium. Furthermore, consider the L1 norm of the gradient:

$$\begin{aligned}
&||\frac{\partial}{\partial t}\mathsf{NLL}(\mathrm{Sm}(t), k)||_1 \qquad (4)\\
=&\sum_i |-\delta_{ik} + \mathrm{Sm}(t, i)|\\
=&\sum_{i\neq k} \mathrm{Sm}(t, i) + (1 - \mathrm{Sm}(t, k))\\
=&2 - 2\cdot\mathrm{Sm}(t, k),
\end{aligned}$$

the gradient is larger when the NLL loss is larger, and NLL is not bounded above. If the adversarial objective is to maximize the NLL, it can accelerate towards infinity, which causes strong instability.

Remember that a basic trick in vanilla GAN is that instead of letting the generator maximize

$$-\ln(1 - D(G(z))), \qquad (5)$$

we let it minimize

$$-\ln(D(G(z))). \qquad (6)$$

In a similar vein, instead of maximizing

$$\mathsf{NLL}(p, k) = -\ln p_{(k)}, \qquad (7)$$

we can minimize what we call "negative log unlikelihood"

$$\mathsf{NLU}(p, k) = -\ln(1 - p_{(k)}). \qquad (8)$$

The derivatives are computed as:

$$\begin{aligned}
&\frac{\partial}{\partial t_{(k)}}\mathsf{NLU}(\mathrm{Sm}(t), k) \qquad (9)\\
=&\frac{\partial}{\partial t_{(k)}} - \ln(1 - \frac{e^{t_{(k)}}}{\sum_j e^{t_{(j)}}})\\
=&-\frac{\sum_j e^{t_{(j)}}}{\sum_j e^{t_{(j)}} - e^{t_{(k)}}} \cdot -\frac{e^{t_{(k)}}(\sum_j e^{t_{(j)}} - e^{t_{(k)}})}{(\sum_j e^{t_{(j)}})^2}\\
=&\mathrm{Sm(t, k)},
\end{aligned}$$

$$\frac{\partial}{\partial t_{(i)}}\mathsf{NLU}(\mathrm{Sm}(t),k) \quad (i\neq k) \tag{10}$$
$$=\frac{\partial}{\partial t_{(i)}} - \ln(1-\frac{e^{t_{(k)}}}{\sum_j e^{t_{(j)}}})$$
$$= -\frac{\sum_j e^{t_{(j)}}}{\sum_j e^{t_{(j)}} - e^{t_{(k)}}} \cdot -\frac{-e^{t_{(i)}}e^{t_{(k)}}}{(\sum_j e^{t_{(j)}})^2}$$
$$= -\frac{\mathrm{Sm}(t,k)\cdot\mathrm{Sm}(t,i)}{1-\mathrm{Sm}(t,k)}.$$

At the assumed equilibrium $\mathrm{Sm}(t)=q$, where $q$ is the class frequency in the dataset, these evaluate to

$$\frac{\partial}{\partial t_{(k)}}\mathsf{NLU}(\mathrm{Sm}(t),k)\Big|_{\mathrm{Sm}(t)=q} \tag{11}$$
$$=q_{(k)},$$

$$\frac{\partial}{\partial t_{(i)}}\mathsf{NLU}(\mathrm{Sm}(t),k)\Big|_{\mathrm{Sm}(t)=q} \quad (i\neq k) \tag{12}$$
$$= -\frac{q_{(k)}\cdot q_{(i)}}{1-q_{(k)}}.$$

If the classes are not evenly distributed in the dataset, this may not satisfy the condition that the assumed equilibrium is a stationary point of $\sum_k q_{(k)}\cdot\mathsf{NLU}(\mathrm{Sm}(t),k)$. To achieve this, we need to properly weight the NLU by class. We can do this by scaling Equation 11 and Equation 12 to match Equation 1. We define the weighted negative log unlikelihood function as:

$$\mathsf{NLU}_q(p,k) = -\frac{1-q_{(k)}}{q_{(k)}}\ln(1-p_{(k)}). \tag{13}$$

Then we have, at the assumed equilibrium:

$$\frac{\partial}{\partial t_{(i)}}\sum_k q_{(k)}\cdot\mathsf{NLU}_q(\mathrm{Sm}(t),k)\Big|_{\mathrm{Sm}(t)=q} \tag{14}$$
$$=\sum_{k\neq i} -q_{(k)}\cdot\frac{1-q_{(k)}}{q_{(k)}}\cdot\frac{q_{(k)}\cdot q_{(i)}}{1-q_{(k)}} + q_{(i)}\cdot\frac{1-q_{(i)}}{q_{(i)}}\cdot q_{(i)}$$
$$=\sum_{k\neq i} -q_{(k)}q_{(i)} + q_{(i)}(1-q_{(i)})$$
$$= -(1-q_{(i)})q_{(i)} + q_{(i)}(1-q_{(i)})$$
$$=0.$$

And the L1 norm of the gradient would be:

$$||\frac{\partial}{\partial t}\mathsf{NLU}_q(\mathrm{Sm}(t),k)||_1 \tag{15}$$
$$=\sum_i \frac{1-q_{(k)}}{q_{(k)}}\cdot\mathrm{Sm}(t,k)\left(1+\sum_{i\neq k}\frac{\mathrm{Sm}(t,i)}{1-\mathrm{Sm}(t,k)}\right)$$
$$=2\cdot\frac{1-q_{(k)}}{q_{(k)}}\cdot\mathrm{Sm}(t,k),$$

which equals to Equation 4 at the assumed equilibrium, and has the desired property that a smaller value of $\mathrm{Sm}(t,k)$ gives a smaller gradient. Evaluating the second derivative at the assumed equilibrium gives

$$\frac{\partial^2}{\partial t_{(k)}^2}\mathsf{NLU}(\mathrm{Sm}(t),k)\Big|_{\mathrm{Sm}(t)=q} \tag{16}$$
$$=q_{(k)}(1-q_{(k)}),$$

$$\frac{\partial^2}{\partial t_{(i)}^2}\mathsf{NLU}(\mathrm{Sm}(t),k)\Big|_{\mathrm{Sm}(t)=q} \quad (i\neq k) \tag{17}$$
$$=\frac{q_{(k)}q_{(i)}(2q_{(i)}+q_{(k)}-1)}{(1-q_{(k)})^2},$$

$$\frac{\partial^2}{\partial t_{(i)}^2}\sum_k q_{(k)}\mathsf{NLU}_q(\mathrm{Sm}(t),k)\Big|_{\mathrm{Sm}(t)=q} \tag{18}$$
$$=q_{(i)}(1-q_{(i)})^2 + \sum_{k\neq i}\frac{q_{(k)}q_{(i)}(2q_{(i)}+q_{(k)}-1)}{1-q_{(k)}}$$
$$=q_{(i)}\left((1-q_{(i)})^2 + \sum_{k\neq i}(\frac{2q_{(k)}q_{(i)}}{1-q_{(k)}} - q_{(k)})\right)$$
$$>q_{(i)}\left((1-q_{(i)})^2 + \sum_{k\neq i} q_{(k)}(2q_{(i)}-1)\right)$$
$$=q_{(i)}\left((1-q_{(i)})^2 + (1-q_{(i)})(2q_{(i)}-1)\right)$$
$$=q_{(i)}(1-q_{(i)})(1-q_{(i)}+2q_{(i)}-1)$$
$$=q_{(i)}^2(1-q_{(i)})$$
$$>0.$$

So the assumed equilibrium is indeed a local minimum of the adversarial loss function we want to minimize.

## 1.2. Sample-Space Classification

The Stage I training procedure of generating samples from random labels for classification is not straightforward to understand, and here we give an explanation.

The distribution of the encoder's output in the code space has few constraints, and there can be different networks that give very different distribution in the code space but are nevertheless essentially equivalent. For example, assume that the last layer of the encoder and the first layer of the generator are linear and the dimension of the code space is $d$. Then we can take an invertible $d\times d$ matrix $M$. We multiply the last layer weights of the encoder by $M$ on the right and multiply the first layer weights of the generator by $M^{-1}$ on the left. In terms of reconstruction, the modified network

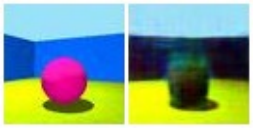


Figure 1: Ill-formed sample generated from mean labeled code.

gives the exact same result as the original, but the code distribution in the code space has been transformed.

This becomes a problem with adversarial training: if the classifier operates in the code space, then to avoid being successfully classified, instead of removing information about the labeled factors from its output, the unknown factor encoder can change its output distribution to confuse the classifier, which results in the code distribution fluctuating constantly in the code space.

In contrast, in the sample space, the distribution of generated samples is anchored to the distribution of training samples and cannot change freely. So, operating the classifier in the sample space can potentially reduce fluctuation in the code space and improve stability. Then the question is which samples should be the input to the classifier.

The reconstructed sample cannot serve as the input to the classifier, since it must contain full information of the input sample, including those about the labeled factors, which is in conflict with the adversarial objective of making the classifier unable to classify by the labeled factor.

Another choice is to combine the unknown code with some kind of "neutral" labeled code, for example, the mean of all label embeddings. The problem is that the "mean" labeled code may not be "typical": as can be seen in Figure 4, in the *3D Shapes* dataset, the network learns that the ten classes of each color attribute are arranged like a circle, but no samples are distributed near the center of the circle. In this case, the mean labeled code does not produce a well-formed sample. An example is shown in Figure 1: on the left is the input. The floor hue is the unknown factor. In the generated sample on the right, all labeled factors have been replaced by the mean embedding, and the generated sample is ill-formed.

The result is that the encoder can encode labeled information about the input sample without being detected: the generator can easily recognize an invalid mean labeled code, and when it receives one it generates ill-formed samples so that the classifier cannot classify the generated sample, regardless of what the encoder has encoded.

So the unknown code used for generating the input to the classifier must be a typical code but at the same time independent from the input sample. Thus, we use the embedding of a random label chosen independently from the input sample.

### 1.3. Continuous-Valued Factor Disentanglement

When presenting our method, we assumed that all labels are discrete, class-type labels. Here we discuss the treatment of continuous-valued factors.

Continuous-valued labels are usually associated with regression problems. So it is reasonable to first attempt to use regressors in place of the classifiers. But there are obvious problems with this approach. Consider training samples $x^{(i)}$ each associated with a single, real-valued label $y^{(i)}$. For each $x^{(i)}$, compute $\overline{x}^{(i)\prime}$ as in Equation 3 in the main text. The regressor $C$ should minimize some kind of distance between $C(\overline{x}^{(i)\prime})$ and $y^{(i)}$, say squared distance $(C(\overline{x}^{(i)\prime}) - y^{(i)})^2$. Then, in the assumed equilibrium where the encoder does not encode any information about the labeled factor, the regressor can only make a guess. To minimize the expected loss, the best guess should be the mean of all $y^{(i)}$. Let $y^* = \sum_{i=1}^{n} y^{(i)}$. Then if there exists training sample $x^*$ whose label is $y^*$ or very close to $y^*$, any adversarial training would not guarantee to prevent the encoder from encoding full information of $x^*$: there is no way to distinguish whether the regressor is giving a $y^*$ because it has detected labeled information in its input, or it is giving a $y^*$ because it detected no such information and is making a guess.

While we have not stated explicitly, we have already provided the solution to working with continuous-valued factors: note that in the *3D Chairs* dataset, the rotation angle is not a true category-type factor, but a quantized continuous-valued factor. Treating it as a category-type factor gives satisfactory results. Similarly, for any continuous-valued factor, we can always divide its range into a suitable number of buckets and quantize the factor into discrete labels. Generally, a few dozen buckets would work fine.

## 2. Data and Metrics

### 2.1. Data

The image size and list of factors of the datasets are given in Table 1, with the number of possible values of each factor and the length of code we use for the encoder of that factor.

### 2.2. Metrics

The Mutual Information Gap (MIG) was originally proposed for the unsupervised setting. We made some adjustments to the computation of MIG to suit the weakly-supervised setting and to allow multi-dimensional code spaces for each factor.

Let $N$ be the number of factors, $L_i$ be the (discrete) random variable representing the label of factor $i$ and $X_i$ be the vector-valued random variable representing the output of the encoder for factor $i$, $i = 1, 2, \ldots, N$. Let $X_i^{(k)}$ be the $k$-th entry of $X_i$, which is a real-valued random variable.

Table 1: Datasets used for evaluation and comparison.

| Dataset | Factor | # of Values | Code Size |
|---|---|---|---|
| MNIST | Class | 10 | 10 |
| $28 \times 28 \times 1$ | (Style) | N/A | 64 |
| F-MNIST | Class | 10 | 10 |
| $28 \times 28 \times 1$ | (Style) | N/A | 64 |
| 3D Chairs | Model | 1393 | 512 |
| $128 \times 128 \times 3$ | Elevation | 2 | 2 |
| | Azimuth | 31 | 2 |
| 3D Shapes | Floor hue | 10 | 8 |
| | Wall hue | 10 | 8 |
| | Object hue | 10 | 8 |
| $64 \times 64 \times 3$ | Scale | 8 | 8 |
| | Shape | 4 | 8 |
| | Orientation | 15 | 8 |

The normalized mutual information between $L_i$ and $X_j^{(k)}$ is defined as in [5]:

$$\hat{I}(L_i; X_j^{(k)}) = \frac{I(L_i; X_j^{(k)})}{H(L_i)}. \tag{19}$$

Then, the multi-dimensional mutual information between $L_i$ and $X_j$ is defined by taking the maximum of $\hat{I}(L_i; X_j^{(k)})$ over $k$:

$$\hat{I}(L_i; X_j) = \max_k \hat{I}(L_i; X_j^{(k)}). \tag{20}$$

One might argue that it is mathematically more meaningful to take the normalized mutual information between $L_i$ and the whole $X_j$ instead. But we found that, as the dimensionality of the code space increases, the number of samples required to accurately estimate $H(X_j)$ and $H(X_j|L_i)$ increases exponentially. And when the number of samples is insufficient, even a randomly initialized encoder would be incorrectly computed to have normalized mutual information close to one, which makes the evaluation meaningless. So we have to settle with our current definition.

Then, the MIG is computed as

$$\text{MIG} = \frac{1}{N} \sum_i (\hat{I}(L_i; X_i) - \max_{j \neq i} \hat{I}(L_i; X_j)). \tag{21}$$

As we have noted, in Table 2 in the main text we are only concerned with the disentanglement between the combined unknown factor and each individual labeled factor. To reflect this, in the computation of MIG here, in Equation 21 we only take the average over $i$ where factor $i$ is labeled.

Special procedures were taken to compute the MIG for [8]: the definition of MIG requires the distribution of $X_j$ to have continuous support, for otherwise all the normalized mutual information would be equal to one and the MIG would be zero. The output of [8] is in the form of an exact code, rather than a normal distribution as in VAE-based methods, so the $X_j$'s will have discrete support, making MIG non-applicable. Note that during the training of [8], Gaussian noise with fixed standard deviation was added to the content embedding, which effectively turns discrete codes into a distribution with full support. So in the computation of MIG, we similarly add Gaussian noise with a fixed standard deviation. The standard deviation is chosen using the following procedure: for all possible pairs of $(\sigma_1, \sigma_2)$ where $\sigma_1, \sigma_2 \in \{10^{-5}, 2 \times 10^{-5}, 5 \times 10^{-5}, 10^{-4}, \ldots, 1, 2, 5, 10\}$, we compute MIG by adding $\mathcal{N}(0, \sigma_1^2 I)$ to the content ("unknown factor" in our terminology) code and $\mathcal{N}(0, \sigma_2^2 I)$ to the class ("labeled factor") code. The values of $\sigma_1$ $\sigma_2$ are chosen so that the average MIG under two settings on the *3D chairs* dataset (rotation unknown and model unknown) is maximized. By this we determine that $\sigma_1 = 0.05$ and $\sigma_2 = 0.02$.

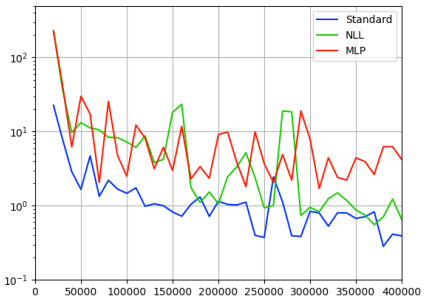



Figure 2: Average squared code distance.

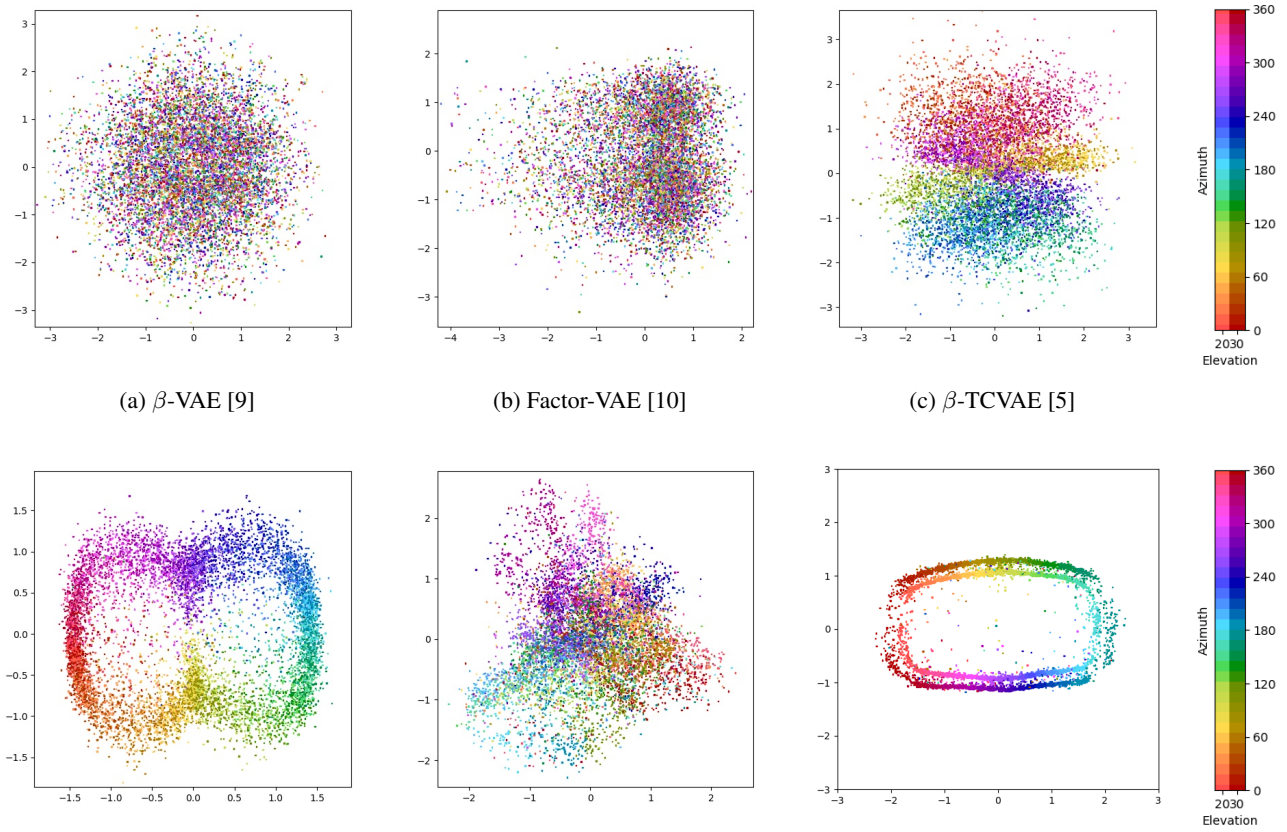
Table 2: Comparison of result with different Stage II configurations on MNIST.

| Configuration | MSE ↓ | MIG ↑ |
|---|---|---|
| Standard | **0.0086** | **0.978** |
| Non-adversarial $R$ | 0.0103 | 0.916 |
| No unknown code condition | 0.0402 | 0.930 |

## 3. Ablation Analysis

### 3.1. Stability of Stage I

Our proposed methods of Negative Log Unlikelihood and sample-space classification aim to improve the stability of encoder-classifier adversarial training. Here we evaluate the effectiveness of these two schemes. Specifically, we track the change of code distribution. We take snapshots of the network at fixed intervals during training. The whole test dataset is encoded, and we compute the average squared distance in the code space, from the code of each sample to




(a) $\beta$-VAE [9] (b) Factor-VAE [10] (c) $\beta$-TCVAE [5]

(d) Pairwise-VAE [4] (e) Lord [8] (f) Ours

Figure 3: Visualizing the disentanglement with test sample distributions.

the code of the same sample in the previous snapshot. In stable training, the encoder should keep the distance small while still finding a good distribution.

We train three variants of Stage I on the 3D chairs dataset with rotation unknown: one standard variant (proposed), one where the adversarial objective is maximizing the NLL loss of the classifier, and one where the classifier is an MLP operating in the code space, with four hidden layers of size 512.

The code distribution is computed every 10,000 iterations until iteration 400,000. The average squared code distance in the unknown code space every 10,000 iterations apart is shown in Figure 2, in logarithm scale. We can see that both NLU and sample-space classification contributed to reducing the fluctuation in code space.

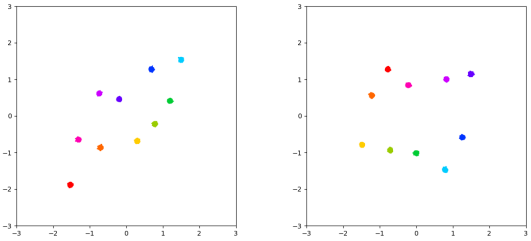
### 3.2. Adversarial Classifier and Condition on Unlabeled Code in Stage II

We evaluate the effectiveness of adversarial classifiers in Stage II compared to non-adversarial ones and examine the necessity of the code distance loss term. We train three variants of Stage II on the *MNIST* dataset: one standard variant (proposed), one without NLU term (remove NLU term from Equation 5e in the main text) so that the classifiers do not try to distinguish generated samples from real ones in the same class, and one without code distance term (remove code distance from Equation 5g in the main text) so that the network does not explicitly preserve the unknown factor.

The initial network weights of $E$ and $G$ for three configurations are inherited from the same Stage I training run so that the result is only affected by Stage II training. We compute the mean squared reconstruction loss and mutual information gap for the final models in Table 2. By using the adversarial Stage II classifier and adding the code distance term, we are able to improve both disentanglement (MIG) and reconstruction (MSE).

## 4. Effect of Factor Correlation

In the datasets used for evaluation, the factors are independent of each other. In particular, in the 3D Shapes and the 3D Chairs datasets, every combination of labels occurred exactly once. In this section we would like to explore the behavior of our method when some of the factors are



(a) Floor hue (b) Wall hue

Figure 4: Distribution of test samples in the code space of the two correlated labeled factors.

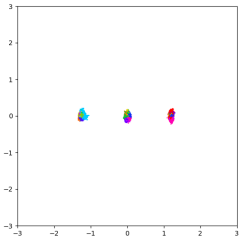


Figure 5: Distribution of test samples in the code space of the unknown encoder when the unknown factor and the labeled factors are correlated.

correlated. For better control, we construct a dataset where the correlation is exactly known: in the 3D Shapes dataset, each of the three color factors has 10 possible values, numbered 0 to 9. We take the subset of the dataset consisting of all images whose *floor hue* and *wall hue* differ by 0 or $\pm 1$, modulo 10.

### 4.1. Correlation Between Two Labeled Factors

The first case is when correlation exists between two of the labeled factors. The semantics of the labeled factors in our networks is enforced to strictly follow the semantics of the labels, so it is expected that our network would behave in the same way as when the labeled factors did not have correlation.

We train our model on the correlated subset with *object hue* being the unknown factor, so that the two correlated factors are both labeled. The distribution of test samples in the two correlated factors is shown in Figure 4. As can be seen, our network successfully learns to encode *floor hue* and *wall hue* as labeled.

While the behavior of the network remains the same, the MIG decreases: due to correlation the mutual information between *floor hue* and *wall hue* is now $\ln 10 - \ln 3$ instead of 0, and a perfect set of encoders would produce an MIG of

$$\frac{1}{6}\left(2\left(1-\frac{\ln 10-\ln 3}{\ln 10}\right)+4\right)\approx 0.8257 \qquad (22)$$

In comparison, our method gives an MIG of 0.8026.

### 4.2. Correlation Between Labeled and Unknown Factors

The situation is more complicated when there is correlation between the unlabeled factor and the labeled factors. Our goal is for the unknown encoder to not encode any information about the labeled factors, which is to say, the conditional distribution of the unknown encoder, given the labeled factors, should be the same regardless of the value of the labeled factors. We train our model with *floor hue* being the unknown factor. In this case, since for any value of *wall hue* there are exactly three possible values of *floor hue*, it can be predicted that our unknown encoder should discover a factor with three discrete values, such that for any *wall hue*, each of the three *floor hues* is encoded as a different value. Note that this discovered factor is not necessarily the "hue difference" of the floor and the wall: there is no guarantee that the three values of this factor correspond to the hue difference being $-1$, 0 and 1 consistently, independent of other factors. The mapping from hue difference to the value of the discovered factor can vary according to *wall hue*. The only guarantee is that for the same *wall hue*, different *floor hues* correspond to different values of the discovered factor.

The distribution of the samples in the unknown encoder's code space, colored by *floor hue*, is shown in Figure 5. Three clusters can be clearly seen.

In general, we can conclude that if the intended semantics of the unknown factor is correlated to the labeled factors, then the factor discovered by our method would have different semantics. This may or may not be a desirable outcome, but it shows that our method is highly effective in ensuring the independence between the discovered unknown factor and the labeled factors.

## 5. Comparisons

### 5.1. Visualization

In Figure 3, for each of the methods compared, we plot the distribution of test samples of the *3D Chairs* dataset in the code space, projected onto the two dimensions having the largest mutual information with the rotation label. In $\beta$-VAE [9] and Factor-VAE [10], no clear color pattern can be observed. In $\beta$-TCVAE and Lord [8], samples with the same rotation are somewhat close together but there is no meaningful order between different rotations. Pairwise-VAE [4] and our method can arrange the azimuth angle correctly into a ring, but the structure is more clear in our method, and also, we can distinguish two slightly different

(a) $\beta$-VAE [9]

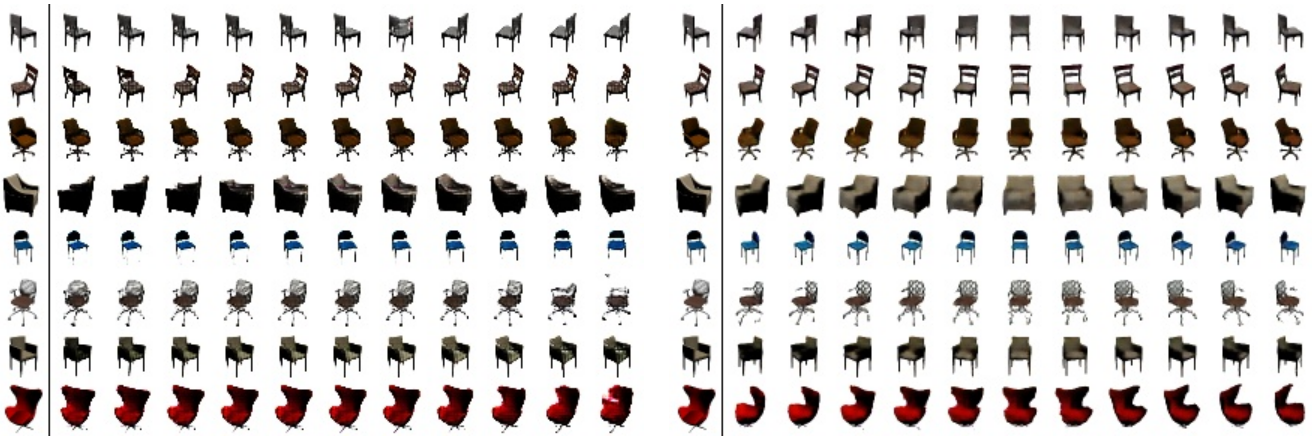


(b) Factor-VAE [10]

(c) $\beta$-TCVAE [5]



(d) Pairwise-VAE [4]



(e) Lord [8]



(f) Ours

Figure 6: Additional results of manipulation comparison on *3D Chairs* by uniformly sampling the latent codes depicting the azimuth rotation. The leftmost column shows the inputs.

elevation angles, which are not distinguished by any other method.

### 5.2. Results

We provide more results of manipulating the latent code related to azimuth rotations and generate the images using different methods in Figure 6.


Figure 7: Additional portrait relighting results.

## 6. Downstream Tasks

### 6.1. Portrait Relighting

In this task, the labeled factor is lighting, represented by the coefficients of spherical harmonics up to second-order, which are 9 real-valued numbers. We show that our disentanglement framework can handle such continuous labels. More portrait relighting results are shown in Figure 7.

### 6.2. Anime Style Transfer

Figure 8 and Figure 9 present more results of anime style transfer generated by fixing either style or content. In Figure 8, we show the styles of the results are consistently faithful to the input. In Figure 9, we explore the diversity of styles our network can learn and demonstrate the ability to generate the same content in different styles where facial shapes, appearances, and aspect ratios are captured.

### 6.3. Landmark-Based Face Reenactment

The landmark-based face reenactment can generate the face motion of one target person from a single image with another source subject's facial expressions and head pose from a driving video. The generated results should preserve the source poses/expressions while matching the target identity. With only the identity label known, we can disentangle unknown poses/expressions from identities and synthesize new landmarks by changing identities in the source landmarks to the target.

We train the landmark-to-image network on the training dataset from VoxCeleb2 [6] which is processed by cropping a $256 \times 256$ face image and extracting its landmarks from each frame. In total, the training data contains $52,112$ videos for $1,000$ randomly selected subjects. We test on a video dataset of $8,000$ frames for $80$ pairwise subjects ($100$ frames per video) randomly sampled from the VoxCeleb2 testing data,

Figure 8: Generated samples with fixed style and random content. Left column shows four input examples of two different artists.

Figure 9: Generated samples with fixed content and random style. The content code is fixed for all results while style codes randomly sample from the style distribution.

| Method | ID ↑ | Pose ↑ | Exp. ↓ |
|---|---|---|---|
| X2face [14] | 0.635 | 0.302 | 0.448 |
| First-order [13] | <u>0.770</u> | 0.822 | 0.274 |
| Ours *w/o* disentanglement | 0.715 | **0.862** | **0.208** |
| Ours *w/* disentanglement | **0.776** | <u>0.840</u> | <u>0.243</u> |

Table 3: Quantitative comparison of methods for cross-subject face reenactment on the VoxCeleb2 testing dataset between our method with and without landmark disentanglement, and two one-shot methods [14], and [13]. Note that our results without disentanglement are generated using the ground truth landmarks from the driving frame so their poses/expressions in the results should be the closest to those in the driving frame, but their identities are very different from the target person. Our method with disentanglement achieves the best in identity preservation and the second-best in poses/expressions.

We present more qualitative comparison in Figure 10 between with and without landmark disentanglement and in Figure 11 against two of the state-of-the-art one-shot face reenactment methods, *i.e.* X2face [14] and First-order [13]. Figure 10 compares the results using source landmarks without disentanglement and synthesized landmarks with disentanglement and shows that landmark disentanglement leads to better identity preservation. Figure 11 shows quantitative comparisons with two one-shot face reenactment baselines and demonstrates our method can better generalize to unseen subjects with more consistent quality under a large variety of poses/expressions.

Besides, we conduct a quantitative comparison as shown in Table 3 to measure the accuracy of disentanglement. We evaluate the results using three metrics (*ID*, *Pose*, and *Exp.*) for identity, pose, and expression respectively. *ID* measures the identity similarity between the resulting face and the target person by computing cosine similarity between their embedding vectors of the face recognition network VGGFace2 [3]. *Pose* measures the similarity between the resulting head pose and the source subject's pose in the driving frame by computing cosine similarity between their head rotations in radians around the X, Y, and Z axes estimated by OpenFace [2]. *Exp.* measures the difference between the resulting expression and the source expression in the driving frame by computing the L2 distance of their intensities of corresponding facial action units detected by OpenFace. Note that our results generated without disentanglement directly use the landmarks from the driving frame so their poses/expressions in the results are the closest to those in the driving frame but their identities mismatch the target person. In contrast, our method with disentanglement largely increases the identity accuracy while achieving comparable accuracy in poses/expressions.

Table 4: Quantitative comparison with the state-of-the-art methods for skeleton disentanglement on Mixamo.

| Method | MSE ↓ | MAE ↓ |
|---|---|---|
| [15] | 0.0131 | 0.0673 |
| [1] | 0.0151 | 0.0749 |
| Ours | **0.0056** | **0.0498** |

### 6.4. Skeleton-Based Body Motion Retargeting

We train our skeleton disentanglement network on the dataset of Mixamo [11]. It contains synthetic 3D skeletons of approximately 800 unique motion sequences each of 36 distinct characters ground truth labels for motion, view, and pose. We follow the same setting in [15] to use 32 of these characters with 800 sequences of each one for training and the rest for testing. There is no overlap in motion sequence and character between training and testing. In training, we project the 3D joint coordinates on-the-fly onto 2D with viewing angles chosen randomly. The labeled factors are the identity of the character and the view angle and the unknown factor is the motion. The network sees only one frame at a time instead of a motion sequence.

In Table 4, we quantitatively compare with the state-of-the-art methods ( *i.e.*, [15] and [1]), which are task-specific, focusing on skeleton disentanglement. The method of [15] is unsupervised while the method of [1] utilizes full supervision. We perform evaluations on the same held-out test set from Mixamo (with ground truth available) using MSE and MAE as the metrics, reported in the original scale of the data. Our method, using weak supervision, outperforms both methods in terms of numerical joint position error but does not rely on any domain-specific prior knowledge.

Figure 12 shows that we can independently control identity, view and motion, synthesizing 2D skeletons with novel identity, view and motion while preserving the remaining factors unchanged.

Figure 13 shows more results of 2D skeleton-based motion retargeting. Although driven by different subjects, the target skeleton identity is completely preserved in the generated results.

## References


[1] Kfir Aberman, Rundi Wu, Dani Lischinski, Baoquan Chen, and Daniel Cohen-Or. Learning character-agnostic motion for motion retargeting in 2d. *ACM Trans. Graph.*, 38(4):75:1–75:14, 2019.

[2] Tadas Baltrusaitis, Amir Zadeh, Yao Chong Lim, and Louis-Philippe Morency. Openface 2.0: Facial behavior analysis toolkit. In *13th IEEE International Conference on Automatic Face & Gesture Recognition, FG 2018, Xi'an, China, May 15-19, 2018*, pages 59–66. IEEE Computer Society, 2018.

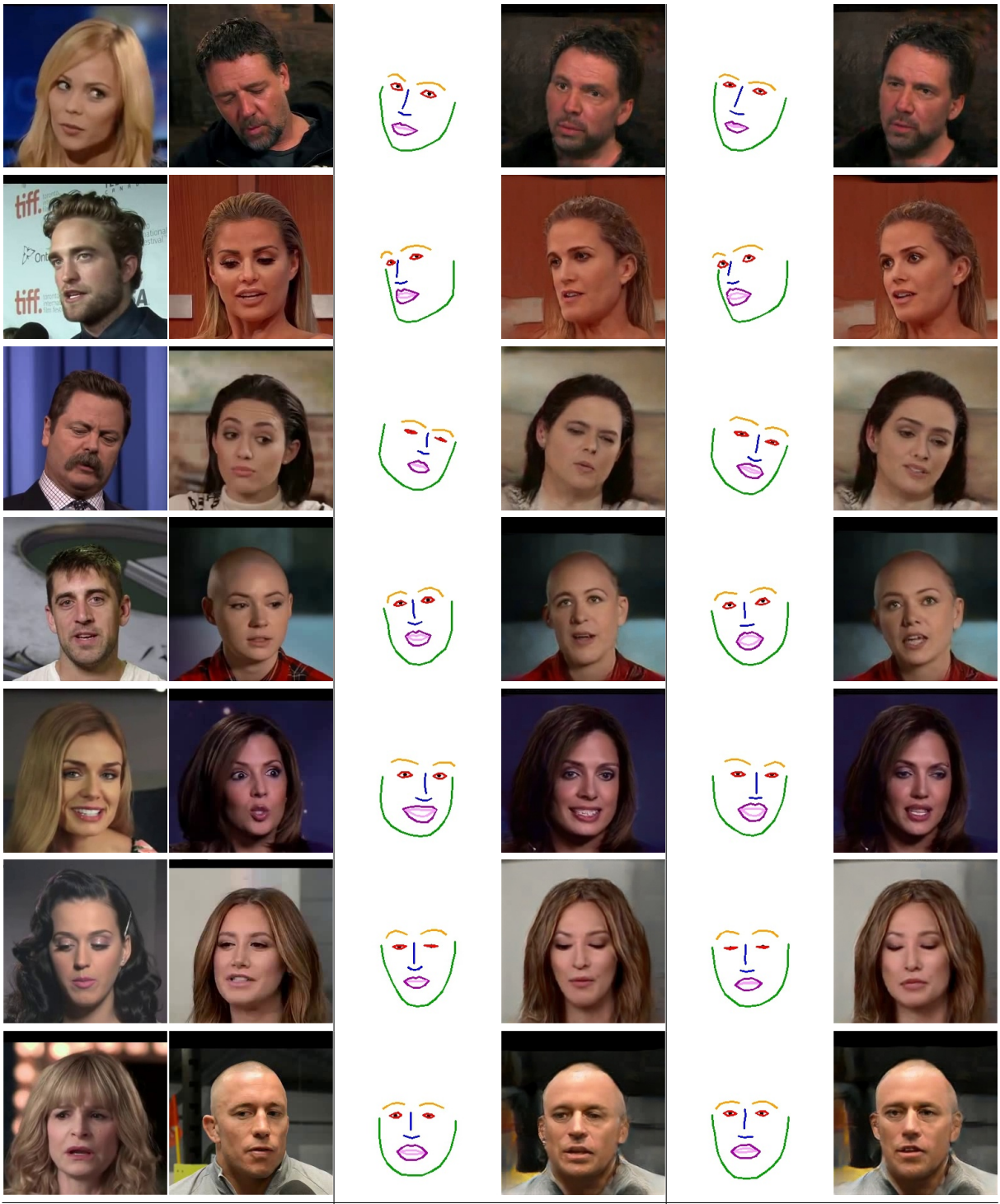



Figure 10: Qualitative comparison on face image reenactment between the translation results without (*w/o*) and with (*w/*) facial landmark disentanglement.

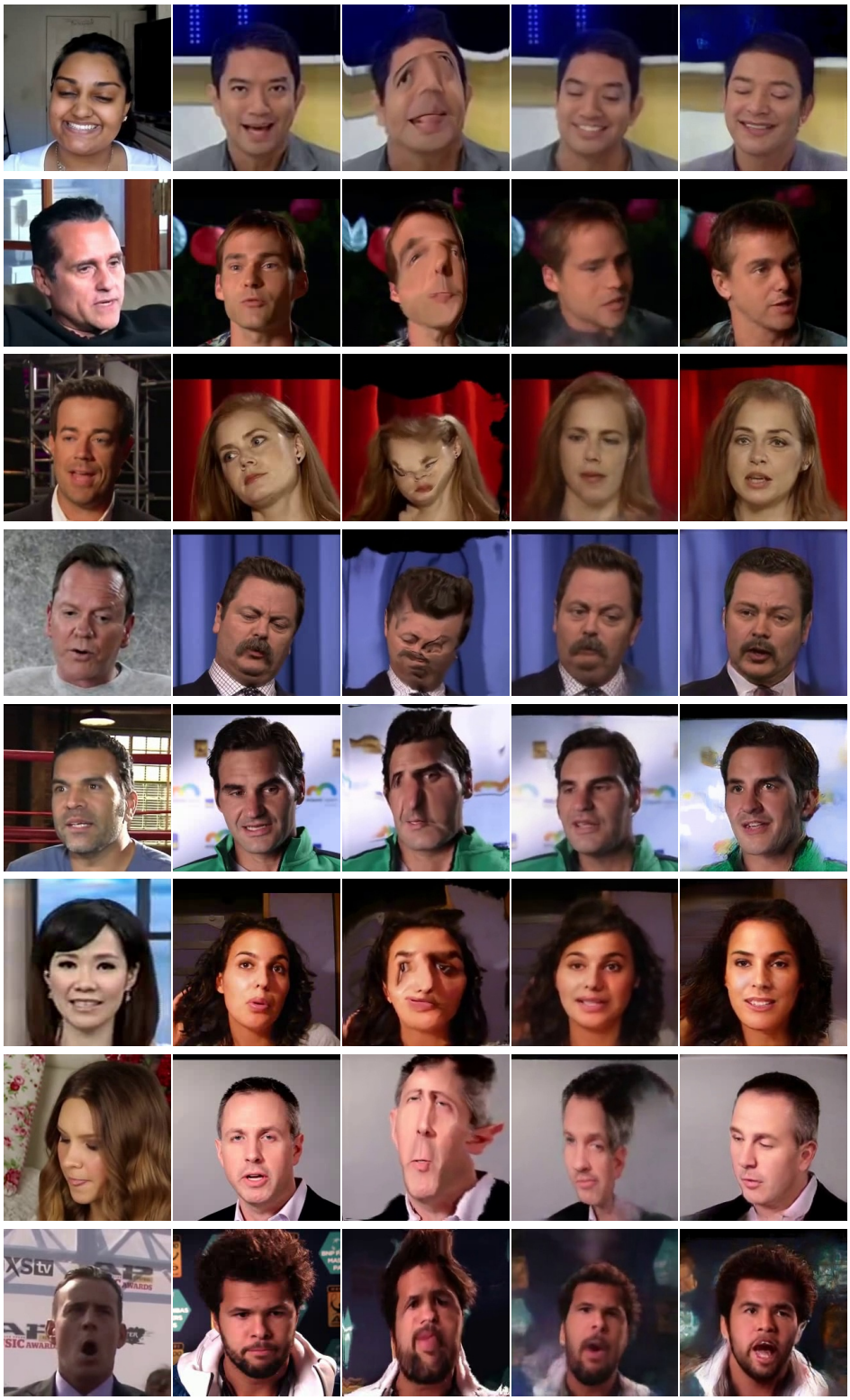
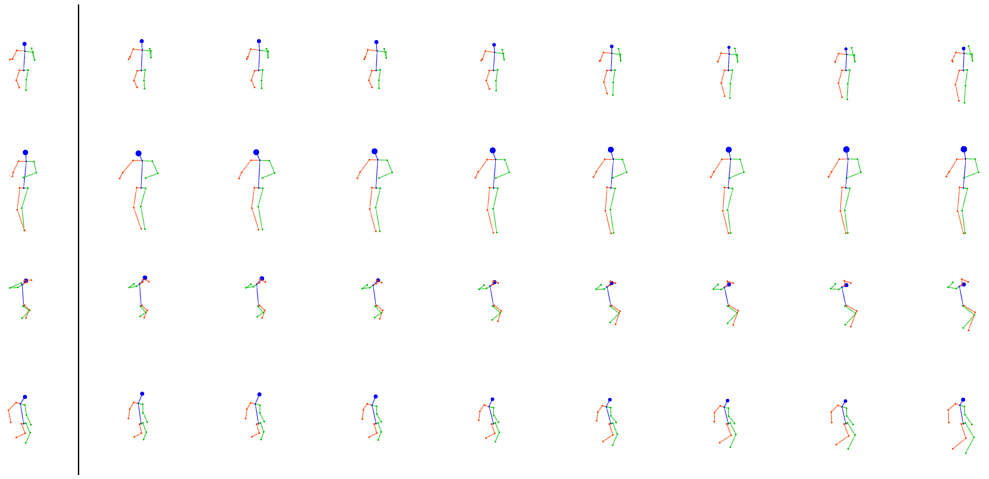
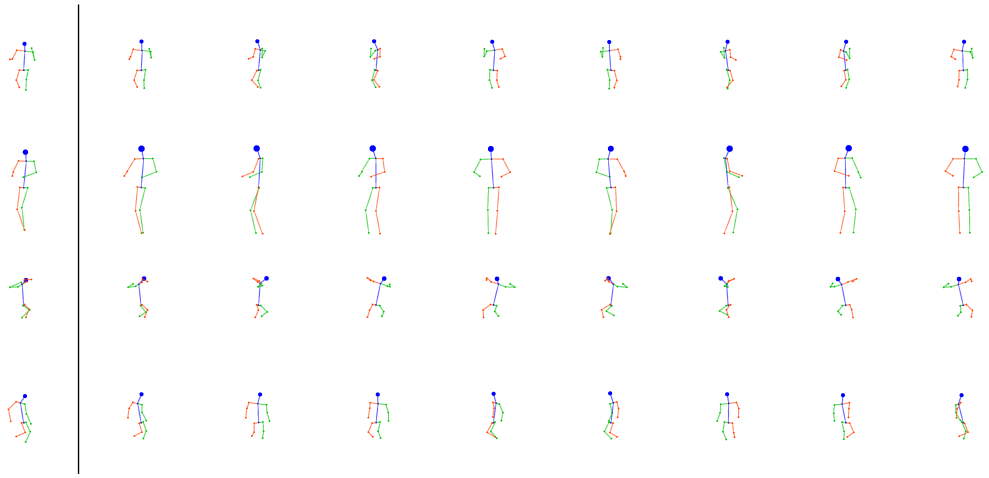


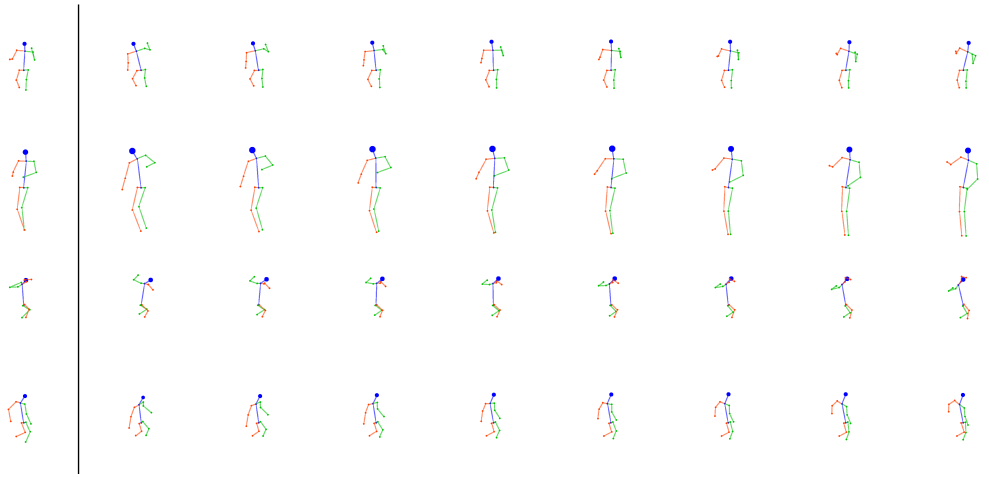

Figure 11: Qualitative comparison on face image reenactment between our method and face motion transfer networks: X2face [14], and First-order [13] using images from Voxceleb2 [7] and FaceForensics++ [12].



(a) Identity synthesis with view and motion fixed

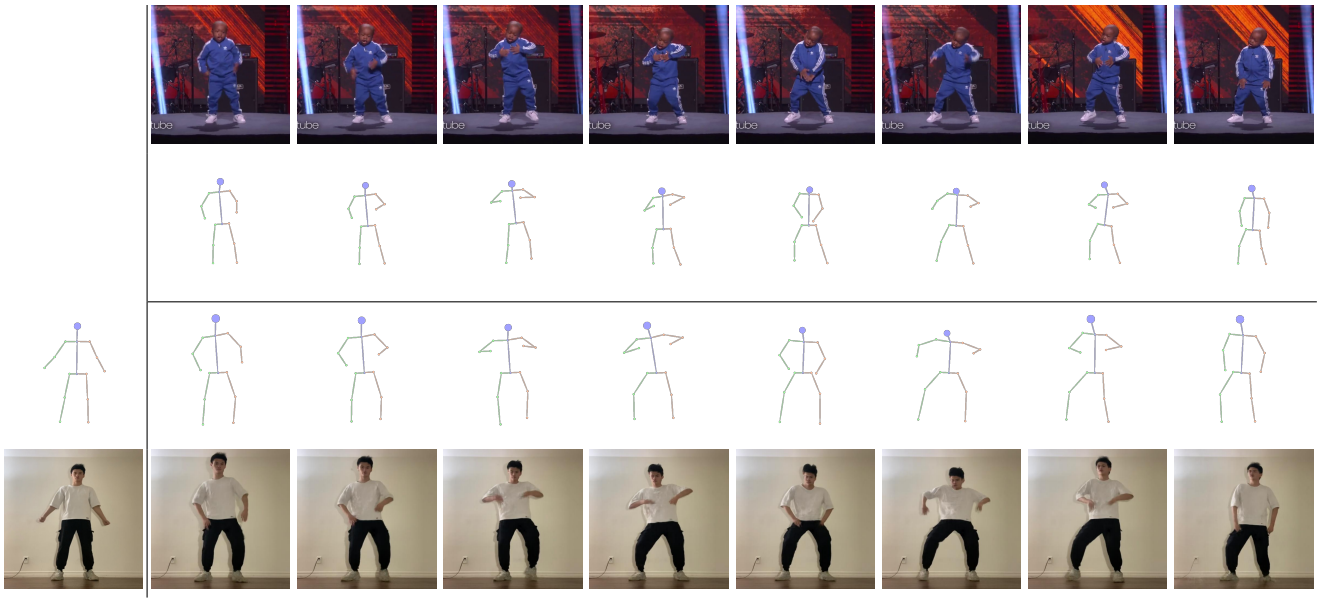


(b) View synthesis with identity and motion fixed

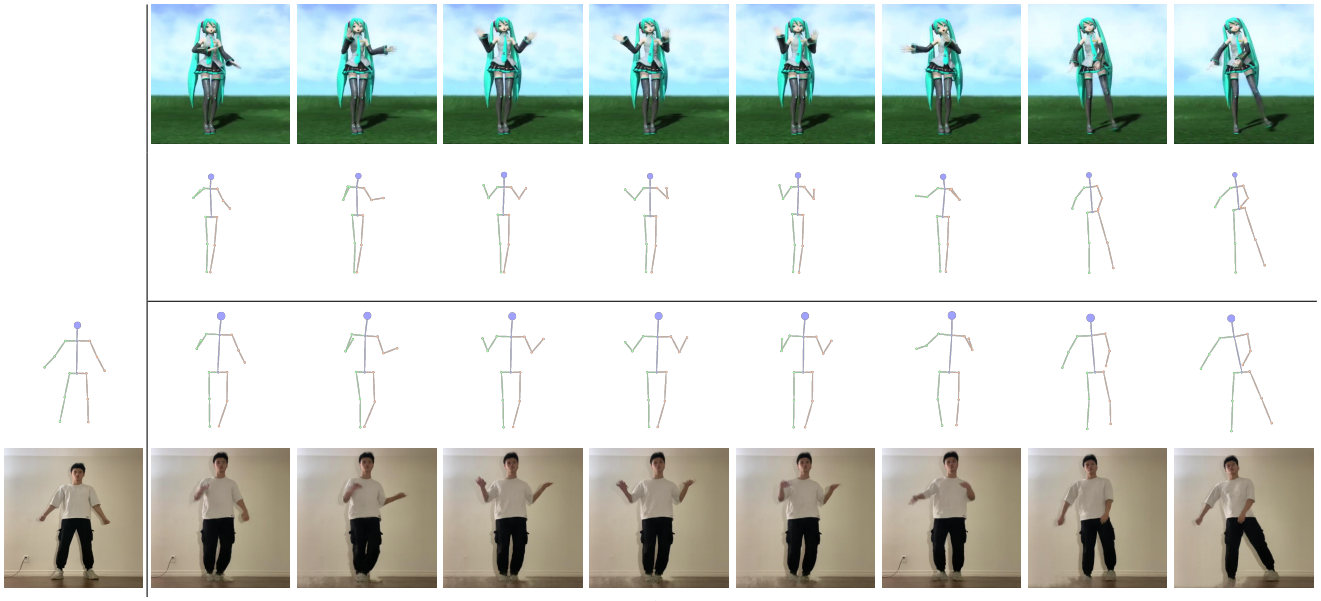


(c) Motion synthesis with identity and view fixed

Figure 12: Novel synthesis of identity, view and motion. Left column shows the input and the rest represents the generated skeletons by individually controlling identity, view and motion while fixing the others.



(a)



(b)


[3] Qiong Cao, Li Shen, Weidi Xie, Omkar M. Parkhi, and Andrew Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *13th IEEE International Conference on Automatic Face & Gesture Recognition, FG 2018, Xi'an, China, May 15-19, 2018*, pages 67–74. IEEE Computer Society, 2018.

[4] Junxiang Chen and Kayhan Batmanghelich. Weakly supervised disentanglement by pairwise similarities. In *AAAI 2020*, pages 3495–3502, 2020.

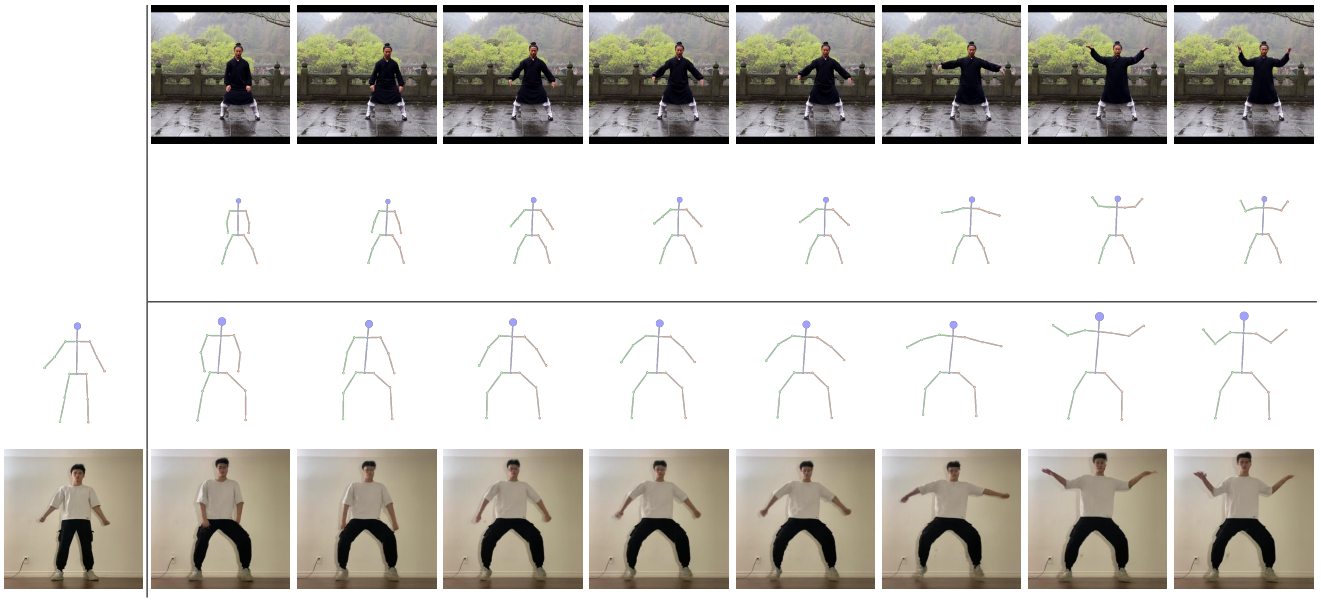
[5] Tian Qi Chen, Xuechen Li, Roger B. Grosse, and David Duvenaud. Isolating sources of disentanglement in variational autoencoders. In *NeurIPS 2018*, pages 2615–2625, 2018.

[6] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. Voxceleb2: Deep speaker recognition. In B. Yegnanarayana, editor, *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018*, pages 1086–1090. ISCA, 2018.

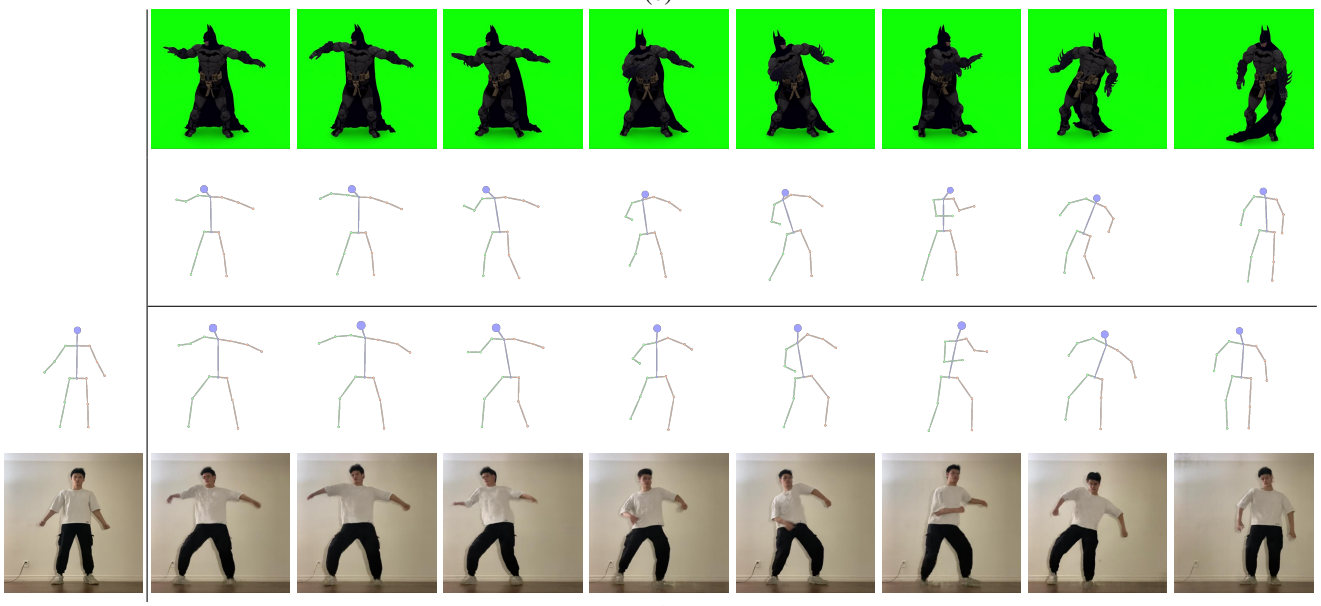
[7] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. VoxCeleb2: Deep Speaker Recognition. In *Interspeech 2018*, pages 1086–1090, 2018.

[8] Aviv Gabbay and Yedid Hoshen. Demystifying inter-class disentanglement. In *ICLR 2020*, 2020.

[9] Irina Higgins, Loïc Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *ICLR*



(c)



(d)

Figure 13: Motion retargeting results from different driving subjects to the same target person. In each case, left column shows the target person and his skeleton, right columns (from top to bottom) represent input source frames, extracted source skeletons, transformed skeletons, and generated target frames.


*2017*, 2017.

[10] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. In *ICML 2018*, volume 80, pages 2654–2663, 2018.

[11] Mixamo. Mixamo. `https://www.mixamo.com/`.

[12] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics++: Learning to detect manipulated facial images. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 1–11. IEEE, 2019.

[13] Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, and Nicu Sebe. First order motion model for image animation. In *NeurIPS 2019*, pages 7135–7145, 2019.

[14] Olivia Wiles, A. Sophia Koepke, and Andrew Zisserman. X2face: A network for controlling face generation using images, audio, and pose codes. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference,*

*Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*, volume 11217 of *Lecture Notes in Computer Science*, pages 690–706. Springer, 2018.

[15] Zhuoqian Yang, Wentao Zhu, Wayne Wu, Chen Qian, Qiang Zhou, Bolei Zhou, and Chen Change Loy. TransMoMo: Invariance-Driven Unsupervised Video Motion Retargeting. In *CVPR 2020*, pages 5305–5314, 2020.