4.How to improve detecting AI voice changer with HuBERT
HuBERT を用いたボイスチェンジャ検出の改良

柴沼 厳 Itsuki Shibanuma　　蓬莱 尚幸 Hisayuki Horai

2025 年 2 月 16 日

1　はじめに

音声のディープフェイクを利用した詐欺、ビデオ会議でのディープフェイクを利用した詐欺、政治家を装ったディープフェイクによる情報操作など近年、ディープフェイク技術を利用した悪用が増加している。音声のディープフェイク技術は、小さなデータセットで、特定の人物の声を再現し相手を騙すことができるため、特に危険である。本研究では、RVC と呼ばれる音声変換技術を用いた AI ボイスチェンジャーの検出における改善手法を提案する。RVC(Retrieval-Based Voice Conversion) は、リアルタイムでの音声変換 (VC: Voice Conversion) が可能な AI ボイスチェンジャーであり、内部で HuBERT というモデルを使用している。本研究における提案手法は、HuBERT を用いた AI ボイスチェンジャーの検出手法であり、HuBERT の最終出力層に、CNN を用いた特徴量を加えることで安定した検出を実現する。

2　HuBERT のモデル構造と数式の詳細解説

2.1　モデル構造の概要

HuBERT は、音声データの高精度な表現学習を目的としており、以下の 2 つの主要なステップを特徴とする：

1. **クラスタリングを用いた目標生成**: 音声データを事前にクラスタリングし、離散的な「隠れ単位」(hidden units) を生成します。例えば、K-means クラスタリングを用いて音声データを 100 または 500 クラスに分類します。
2. **マスクされた予測 (Masked Prediction)**: マスクされた入力音声データを使用し、マスクされた部分の隠れ単位を予測するタスクを学習します。これにより、非マスク領域の情報を基にマスク領域を推測する能力がモデルに学習されます。

HuBERT は、BERT アーキテクチャを応用し、音声データの時間的構造と高次元表現の両方を学習する。[1].

2.2 数式の詳細

2.2.1 クラスタリングを用いた隠れ単位の生成

HuBERT では、音声データ X を T フレームの系列として表現する：

$$X = [x_1, x_2, \ldots, x_T]$$

クラスタリングモデル h を使用して、各フレーム x_t にクラスタラベル z_t を割り当てる：

$$Z = h(X) = [z_1, z_2, \ldots, z_T], \quad z_t \in \{1, \ldots, C\}$$

ここで、C はクラスタ数である（例：$C = 100$）。

2.2.2 マスクされた予測の学習

HuBERT は、以下のようにマスクされた音声データ $\tilde{X}$ を生成する：

$$\tilde{X} = r(X, M)$$

ここで、$M \subseteq \{1, \ldots, T\}$ はマスクされたフレームのインデックス集合、$r(\cdot)$ は対応するフレームをマスク埋め込み $\tilde{x}$ に置き換える操作を表す。

モデル f は、マスクされたデータ $\tilde{X}$ を入力とし、各タイムステップ t の目標分布を予測する：

$$p_f(z_t | \tilde{X}, t)$$

2.2.3 損失関数

HuBERT の損失関数は、マスクされた領域 M と非マスク領域に基づくクロスエントロピー損失の重み付き和として定義される：

$$L = \alpha L_m + (1 - \alpha) L_u$$

- マスクされた領域での損失：

$$L_m(f; X, M, Z) = -\sum_{t \in M} \log p_f(z_t | \tilde{X}, t)$$

- 非マスク領域での損失：

$$L_u(f; X, M, Z) = -\sum_{t \notin M} \log p_f(z_t | \tilde{X}, t)$$

ここで、α は損失関数の重みを調整するハイパーパラメータである。特に、$\alpha = 1$ の場合、損失はマスク領域のみに基づく。

2.3 モデル構成

HuBERT モデルは以下の構成を持つ：

1. **畳み込みエンコーダ**: 音声波形を低次元表現に変換する。
2. **BERT エンコーダ**: Transformer アーキテクチャを用いて、時間的相関をモデル化する。
3. **射影層**: 特徴量をクラスタリングラベルの分布に変換する。

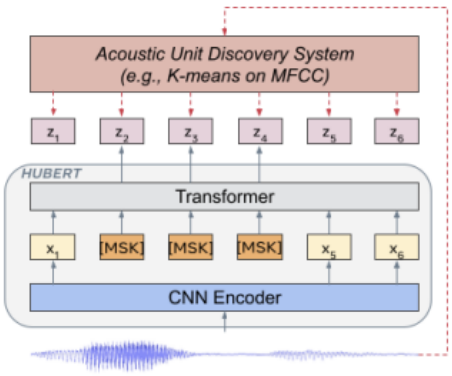

図 1　図 1: HuBERT アプローチは、k-means クラスタリングを 1 回または複数回繰り返すことで生成されたマスクされたフレーム（図中の y_2, y_3, y_4）の隠れクラスタ割り当てを予測する。

3　提案手法

従来手法である、音声データから得られた MFCC を CNN によって分類するモデルでは、90% 以上の精度を達成を達成することはできないが、とても学習コストが低く、ロバストな学習が可能であった。一方、HuBERT を単体で用いたモデルでは、学習コストが高く、ロバストな学習が難しいが、90% 以上の精度を達成することができる。そこで、HuBERT の高い精度と CNN のロバスト性を組み合わせることで、安定した学習と高い精度を両立することができると考えた。

今回私が提案する手法は以下のとおりである。

まず初めに、音声データから得られた MFCC を CNN によって、(1,768) のテンソルに変換する。そうして得られたテンソルを HuBERT の最終出力層と足し合わせる。

次に、このテンソルを Linear 層に入力し、最終的に Softmax 関数によって 2 クラス分類を行う。このような構成を取ることで、HuBERT の高い精度と CNN のロバスト性を組み合わせることができる。

4　実験

4.1　実験設定

RVC を用いて変換された音声データを 500、未変換の音声データを 500、計 1000 のデータセットを作成した。その中から、ランダムサンプリングにより、訓練データとテストデータをそれぞれ 400、100 のデータセットを作成し、実験を行った。実験で使用したモデルは、従来手法である、音声データから得られた MFCC を CNN によって分類するモデル、HuBERT を単体で用いたモデル、提案手法である HuBERT と CNN を組み合わせたモデルの計 3 つである。これらのモデルに対して、入力された音声が RVC を用いて変換されたものか、そうでないかの 2 クラス分類を行うように学習した。

今回の実験において、HuBERT を単体で用いたモデルと、HuBERT と CNN を組み合わせたモデルの学習において、モデルのロバスト性を確認するため、エポック数を 10、20 という比較的大きな値に設定した。

4.2 モデル構成

CNN を用いたモデルの構成は以下の通りである。

```
CNNClassifier(
    (conv1): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
    (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
    (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (dropout_conv): Dropout(p=0.3, inplace=False)
    (fc1): Linear(in_features=10240, out_features=128, bias=True)
    (dropout_fc): Dropout(p=0.5, inplace=False)
    (fc2): Linear(in_features=128, out_features=2, bias=True)
)
```

HuBERT を単体で用いたモデルの構成は以下の通りである。

```
HubertForSequenceClassification(
  (hubert): HubertModel(
    (feature_extractor): HubertFeatureEncoder(
      (conv_layers): ModuleList(
        (0): HubertGroupNormConvLayer(
          (conv): Conv1d(1, 512, kernel_size=(10,), stride=(5,), bias=False)
          (activation): GELUActivation()
          (layer_norm): GroupNorm(512, 512, eps=1e-05, affine=True)
        )
        (1-4): 4 x HubertNoLayerNormConvLayer(
          (conv): Conv1d(512, 512, kernel_size=(3,), stride=(2,), bias=False)
          (activation): GELUActivation()
        )
        (5-6): 2 x HubertNoLayerNormConvLayer(
          (conv): Conv1d(512, 512, kernel_size=(2,), stride=(2,), bias=False)
          (activation): GELUActivation()
        )
      )
    )
    (feature_projection): HubertFeatureProjection(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (projection): Linear(in_features=512, out_features=768, bias=True)
      (dropout): Dropout(p=0.0, inplace=False)
```

```
  )
  (encoder): HubertEncoder(
    (pos_conv_embed): HubertPositionalConvEmbedding(
      (conv): ParametrizedConv1d(
        768, 768, kernel_size=(128,), stride=(1,), padding=(64,), groups=16
        (parametrizations): ModuleDict(
          (weight): ParametrizationList(
            (0): _WeightNorm()
          )
        )
      )
      (padding): HubertSamePadLayer()
      (activation): GELUActivation()
    )
    (layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (layers): ModuleList(
      (0-11): 12 x HubertEncoderLayer(
        (attention): HubertSdpaAttention(
          (k_proj): Linear(in_features=768, out_features=768, bias=True)
          (v_proj): Linear(in_features=768, out_features=768, bias=True)
          (q_proj): Linear(in_features=768, out_features=768, bias=True)
          (out_proj): Linear(in_features=768, out_features=768, bias=True)
        )
        (dropout): Dropout(p=0.1, inplace=False)
        (layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (feed_forward): HubertFeedForward(
          (intermediate_dropout): Dropout(p=0.1, inplace=False)
          (intermediate_dense): Linear(in_features=768, out_features=3072, bias=True)
          (intermediate_act_fn): GELUActivation()
          (output_dense): Linear(in_features=3072, out_features=768, bias=True)
          (output_dropout): Dropout(p=0.1, inplace=False)
        )
        (final_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      )
    )
  )
)
(projector): Linear(in_features=768, out_features=256, bias=True)
(classifier): Linear(in_features=256, out_features=2, bias=True)
```

)

HuBERT と CNN を組み合わせたモデルの構成は以下の通りである。

```
HuBERTWithLogMel(
    (hubert): HubertModel(
        (feature_extractor): HubertFeatureEncoder(
            (conv_layers): ModuleList(
                (0): HubertGroupNormConvLayer(
                    (conv): Conv1d(1, 512, kernel_size=(10,), stride=(5,), bias=False)
                    (activation): GELUActivation()
                    (layer_norm): GroupNorm(512, 512, eps=1e-05, affine=True)
                )
                (1-4): 4 x HubertNoLayerNormConvLayer(
                    (conv): Conv1d(512, 512, kernel_size=(3,), stride=(2,), bias=False)
                    (activation): GELUActivation()
                )
                (5-6): 2 x HubertNoLayerNormConvLayer(
                    (conv): Conv1d(512, 512, kernel_size=(2,), stride=(2,), bias=False)
                    (activation): GELUActivation()
                )
            )
        )
        (feature_projection): HubertFeatureProjection(
            (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
            (projection): Linear(in_features=512, out_features=768, bias=True)
            (dropout): Dropout(p=0.0, inplace=False)
        )
        (encoder): HubertEncoder(
            (pos_conv_embed): HubertPositionalConvEmbedding(
                (conv): ParametrizedConv1d(
                    768, 768, kernel_size=(128,), stride=(1,),
                    padding=(64,), groups=16
                    (parametrizations): ModuleDict(
                        (weight): ParametrizationList(
                            (0): _WeightNorm()
                        )
                    )
                )
                (padding): HubertSamePadLayer()
                (activation): GELUActivation()
```

```
            )
            (layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
            (layers): ModuleList(
                (0-11): 12 x HubertEncoderLayer(
                    (attention): HubertSdpaAttention(
                        (k_proj): Linear(in_features=768, out_features=768, bias=True)
                        (v_proj): Linear(in_features=768, out_features=768, bias=True)
                        (q_proj): Linear(in_features=768, out_features=768, bias=True)
                        (out_proj): Linear(in_features=768, out_features=768, bias=True)
                    )
                    (dropout): Dropout(p=0.1, inplace=False)
                    (layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
                    (feed_forward): HubertFeedForward(
                        (intermediate_dropout): Dropout(p=0.1, inplace=False)
                        (intermediate_dense): Linear(in_features=768, out_features=3072
                        (intermediate_act_fn): GELUActivation()
                        (output_dense): Linear(in_features=3072, out_features=768, bias=
                        (output_dropout): Dropout(p=0.1, inplace=False)
                    )
                    (final_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=
                )
            )
        )
    )
    (cnn): Sequential(
        (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
        (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (4): ReLU()
        (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (fc_mel): Linear(in_features=1228800, out_features=768, bias=True)
    (classifier): Linear(in_features=768, out_features=2, bias=True)
    (loss_fn): CrossEntropyLoss()
)
```

4.3 結果

CNN を用いたモデルによる推論結果は表 1 の通りである。

表 1　CNN を用いたモデルによる推論結果 (epoch=10)

Epoch	Training Loss	Validation Loss	Accuracy (%)
1	0.3538	0.2456	84.00
2	0.1742	0.2422	88.00
3	0.1368	0.2535	80.00
4	0.1663	0.2592	80.00
5	0.1886	0.2644	80.00
6	0.1164	0.2791	80.00
7	0.1543	0.2832	80.00
8	0.2481	0.2811	80.00
9	0.1784	0.2664	80.00
10	0.1607	0.2556	80.00

また、HuBERT を単体で用いたモデルによる推論結果は表 2、表 3 の通りである。

表 2　HuBERT を単体で用いたモデルによる推論結果 (epoch=10)

Epoch	Training Loss	Validation Loss	Accuracy (%)
1	0.698800	0.695777	50.00
2	0.694900	0.694842	50.00
3	0.696400	0.693102	50.00
4	0.690300	0.688122	50.00
5	0.681000	0.665364	100.00
6	0.657100	0.606551	100.00
7	0.549300	0.524111	93.75
8	0.489800	0.403892	93.75
9	0.366900	0.325082	93.75
10	0.262000	0.198047	100.00

HuBERT と CNN を組み合わせてモデルによる推論結果は表 4、表 5 の通りである。

表 3　HuBERT を単体で用いたモデルによる推論結果 (epoch=20)

Epoch	Training Loss	Validation Loss	Accuracy
1	0.698400	0.658245	1.000000
2	0.558400	0.360767	1.000000
3	0.391200	0.156901	1.000000
4	0.129000	0.034920	1.000000
5	0.215400	0.013393	1.000000
6	0.121300	0.007694	1.000000
7	0.005800	0.003924	1.000000
8	0.266900	0.012838	1.000000
9	0.118800	0.007923	1.000000
10	0.007700	0.004464	1.000000
11	0.128900	0.378286	0.941176
12	0.002400	0.421135	0.941176
13	0.001200	0.000653	1.000000
14	1.477500	2.305169	0.411765
15	1.385900	0.685209	0.588235
16	0.658300	0.697818	0.411765
17	0.736900	0.709273	0.411765
18	0.710800	0.688024	0.588235
19	0.726500	0.727446	0.411765
20	0.702600	0.706241	0.411765

4.4　実験の考察

CNN を用いたモデルでは、少ないエッポクで精度が 8 割を超える結果となった。また、エポックを増やしたとしても安定した精度が出るため、ロバストな学習が可能である。しかし、精度が 8 割を超える程度なので、実用性を考えより高い精度を求める場合は、他の手法を用いる必要がある。次に、HuBERT を単体で用いたモデルでは、エポックを増やすにつれて精度と Loss が上下し、安定した学習ができない。そこで、安定した精度で学習が可能な CNN と不安定だが高い精度を出す HuBERT を組み合わせたモデルを構築した。

結果としては表 3 から見えるように、HuBERT を単体で用いたモデルよりも、精度は少し落ちたもののどのエポックに対しても高い精度を出すことができ、Loss も安定して減少しておりロバストな学習が可能であることがわかった。

5　課題と展望

本研究では、オープンソースかつ少ないデータセットで高精度に学習元と同じ声のボイスチェンジャーを実現できる RVC を用いた。しかし、SO-VITS-SVC[2]. や YourTTS[3]. などよりモダンで高精度な AI ボイスチェンジャーが複数存在し、それらに対しても包括的かつ高精度に検出できるような研究が求められる。

表 4　HuBERT と CNN を組み合わせたモデルによる推論結果 (epoch=10)

Epoch	Training Loss	Validation Loss	Accuracy (%)
1	0.684700	0.827347	37.50
2	0.403900	0.329194	87.50
3	0.245500	0.108115	100.00
4	0.145500	0.075007	100.00
5	0.036000	0.099326	93.75
6	0.010700	0.340966	87.50
7	0.005700	0.222607	93.75
8	0.002300	0.067302	93.75
9	0.001000	0.024201	100.00
10	0.000500	0.035744	100.00

参考文献

[1] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, Abdelrahman Mohamed, ”HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units”, arXiv:2106.07447, 2021.

[2] Wen-Chin Huang, Lester Phillip Violeta, Songxiang Liu, Jiatong Shi, Tomoki Toda, ”The Singing Voice Conversion Challenge”,arXiv:2306.14422

[3] Edresson Casanova, Julian Weber, Christopher Shulby, Arnaldo Candido Junior, Eren Gölge, Moacir Antonelli Ponti,”YourTTS: Towards Zero-Shot Multi-Speaker TTS and Zero-Shot Voice Conversion for everyone”,arXiv:2112.02418,2021

表 5　HuBERT と CNN を組み合わせたモデルによる推論結果 (epoch=20)

Epoch	**Training Loss**	**Validation Loss**	**Accuracy**
1	0.694300	0.454093	0.937500
2	0.252600	0.309399	0.937500
3	0.169000	0.217254	0.937500
4	0.080800	0.280794	0.937500
5	0.022100	0.325377	0.750000
6	0.017100	0.343335	0.937500
7	0.021300	0.228357	0.875000
8	0.004500	0.923829	0.937500
9	0.000100	0.160370	0.937500
10	0.000100	0.626578	0.937500
11	0.000300	0.316391	0.937500
12	0.000200	0.630442	0.937500
13	0.000000	0.708360	0.937500
14	0.000000	0.669939	0.937500
15	0.000000	0.631520	0.937500
16	0.000000	0.599144	0.937500
17	0.000000	0.584343	0.937500
18	0.000000	0.575802	0.937500
19	0.000000	0.569419	0.937500
20	0.000000	0.570080	0.937500