Using artificial intelligence to classify English fluency

Classifying English fluency is crucial in English learning, but it requires significant effort from teachers to assess. Therefore, using artificial intelligence to automatically evaluate students' English fluency is extremely useful, providing rapid feedback on their oral practice. This paper primarily uses several commonly used deep learning models to evaluate spoken English fluency. It also attempts to introduce the ViT model, a relatively advanced model in image recognition, to further improve model accuracy. First, we need to build the necessary dataset. Then, we segment the collected audio into 10-second non-repeating audio segments for speech feature extraction. We extract 25 commonly used Mel-spectral coefficients from the audio. Of the 447 audio segments, 70% are used as the training set and 30% as the test set. From these segments, the deep learning models we selected generally achieved a test accuracy of 80%. The test accuracy of the ResNet and ViT models approached 90%.

1. Introduction

Since its inception in the 1950s, artificial intelligence has experienced two booms and busts. In the past decade, with the breakthrough of the bottleneck of computing power and the arrival of the digital age, which provides more data for deep learning, artificial intelligence has entered a period of rapid development. In particular, after the emergence of Chat GPT in 2022 , it quickly became a global hot topic. In October 2023, Gartner released the top ten strategic technology trends in the world for 2024, five of which were related to artificial intelligence. [1]

As one of the three main subjects, English is a subject that almost all Chinese students dedicate a significant amount of time to learning. To improve the effectiveness of English communication with international audiences, we increasingly emphasize improving spoken English. However, we want to explore the use of artificial intelligence technology to quickly distinguish the fluency of English passages.

There is no universal definition of fluency. In fact, each language institution has developed a fluency index based on its internal parameters. In our case, to assess a speaker’s fluency, the fluency index we discuss is mainly based on the absence of unnatural pauses in the voice and speech. If the speech is slow or there are many pauses, it will affect the fluency classification of the speaker. At the same time, fluency and proficiency are different. This means that fluency means that a person can feel comfortable, sound natural, and can manipulate all parts of a sentence at will. [2]

This research primarily uses a deep learning model to differentiate the fluency of spoken English based on audio analysis. First, English spoken audio files are collected and labeled as low or high fluency based on their actual fluency. The specific data collection process is detailed in Chapter 2. Then, the collected audio is segmented into 10-second unique audio segments, and features are extracted from them. The feature vectors of the audio are then extracted, and a deep learning model is used to classify the spoken fluency of the audio. The collected audio is segmented into 10-second unique audio segments, and feature values are extracted. Specific feature values include Mel-frequency cepstral coefficients (MFCCs) and the audio spectrogram . Subsequently, the feature values of the dataset are input into the classification model for training, and its performance is evaluated using test accuracy metrics. We compare three commonly used deep learning models—Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN) —for classification based on MFCC feature values . We also attempt to use the advanced ViT and ResNet models in image recognition for classification based on spectrograms to achieve higher accuracy. For common deep learning models, we selected 25 MFCC Mel-frequency cepstral coefficients (MFCCs) as feature values for audio files. Based on the clear characteristics of fluency in the audio spectrogram — lower fluency audio spectrograms have more discontinuous regions, while higher fluency audio spectrograms have fewer discontinuous regions— we also attempted to use the ViT and ResNet models , which are widely used and highly accurate in image recognition , for fluency classification. Since these models are primarily based on image recognition, we used the spectrogram as the feature value for audio files.

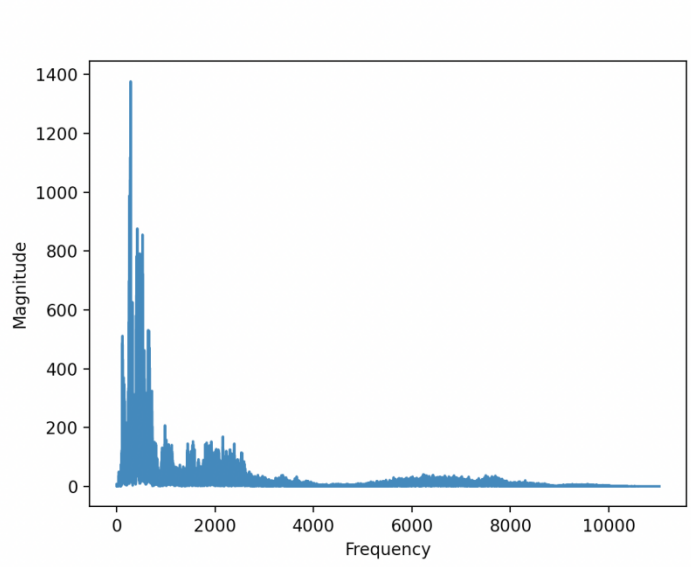
Our final classification experiment data shows that, for English fluency classification, among the three commonly used deep learning models, convolutional neural networks (CNNs) outperform multilayer perceptrons (MLPs) and recurrent neural networks (RNNs). The MLPs achieve nearly 75% accuracy, CNNs

86%, and RNNs 79%. However, ResNet and ViT models based on spectrogram recognition can achieve close to 90% accuracy, slightly better than the three commonly used deep learning models based on MFCC .

2. Prepare the dataset

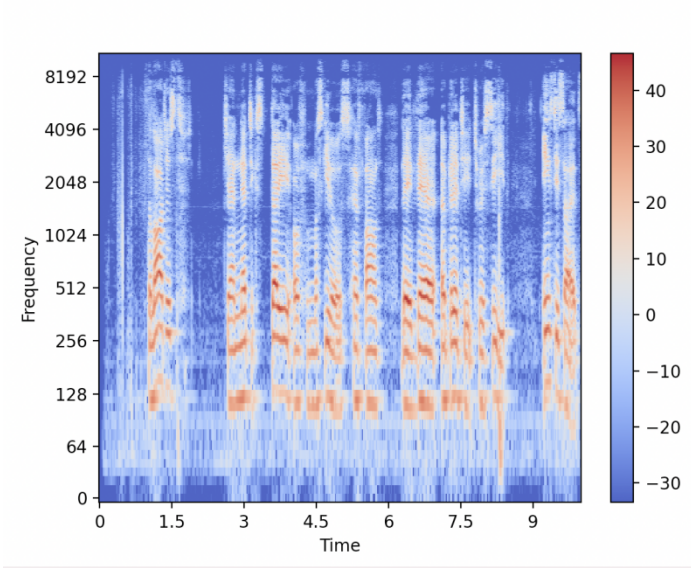
High-quality datasets are essential for many deep learning projects. A major challenge in deep learning model architecture is mastering patterns within the data to achieve sufficient complexity for accurate predictions or classifications. Model accuracy is affected by mislabeled or missing data in the dataset. Furthermore, the model's limitations are related to available computing power and the amount of labeled data required for training. Unfortunately, no publicly available datasets meet the needs of this research. I collected the necessary dataset by having students consciously read fluently and non-fluently. This included labeled English reading audio recordings from 10 male and 5 female students, each approximately one hour long, at different fluency levels (high and low). Audio segmentation is one of the most crucial preprocessing steps in most audio applications. We segmented the collected audio into audio segments and extracted features. In our research, the primary segmentation method was cutting the audio files into 10-second non-overlapping segments. The segmentation of audio segments was mainly performed using a Python library called pydub.

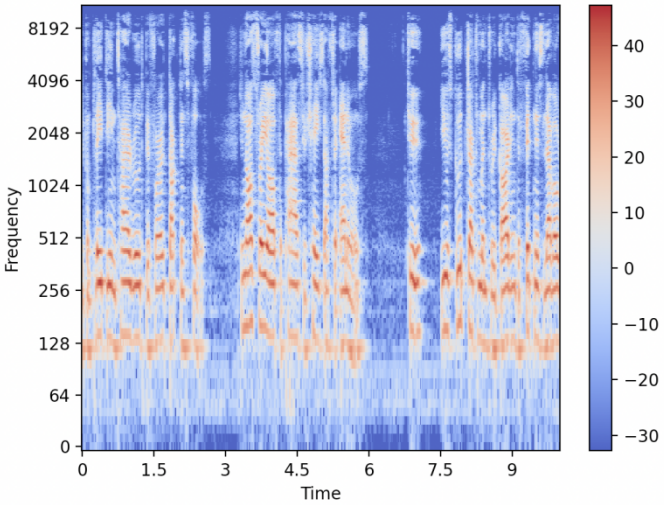
For the feature values of audio segments, we chose to extract Mel frequency cepstral coefficients from the audio. *Mel frequency cepstral coefficients* (MFCC) are a feature widely used in automatic speech and speaker classification because they are suitable for understanding the frequency of human speech. MFCC feature extraction includes two key steps: Mel frequency analysis and cepstral analysis. MFCC is a set of key coefficients used to build the Mel cepstral spectrum. From the segments of the audio signal, we can obtain a set of cepstral spectra that are sufficient to represent the audio signal. Unlike the general cepstral spectrum, the frequency bands on the Mel cepstral spectrum are uniformly distributed on the Mel scale , and such frequency bands will be more linear than we usually see. The cepstral representation is closer to the nonlinear auditory system of humans. Therefore, in this study, we chose Mel cepstral coefficients as the feature extraction method [3] . According to previous studies, 25 Mel cepstral coefficients can provide better results in the field of audio classification, so we chose 25 Mel cepstral coefficients as the input data of three commonly used deep learning models [8] .
We wrote a Python script to perform feature extraction on the created audio clips using a Python package called Librosa (Librosa is commonly used for feature extraction in many functions , allowing the computation of more than 30 audio features). Audio frame $f_0 \ldots , f_n$ therefore have their corresponding eigenvectors $p_0 \ldots , p_n$. Librosa can also use the display.specshow function to draw audio spectrograms. For completeness, we show the following common spectrograms of a single audio segment .

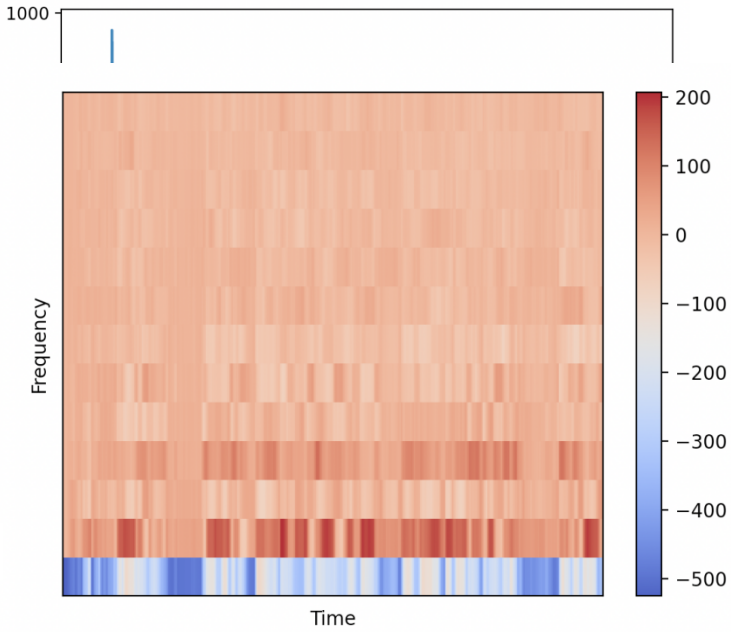


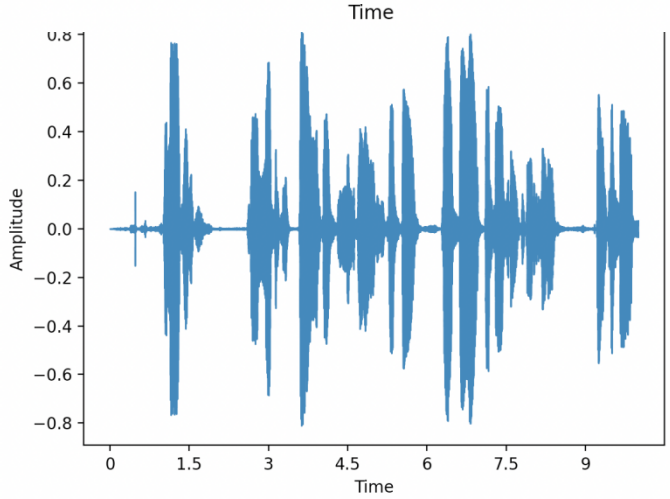
不流利的音频波形图

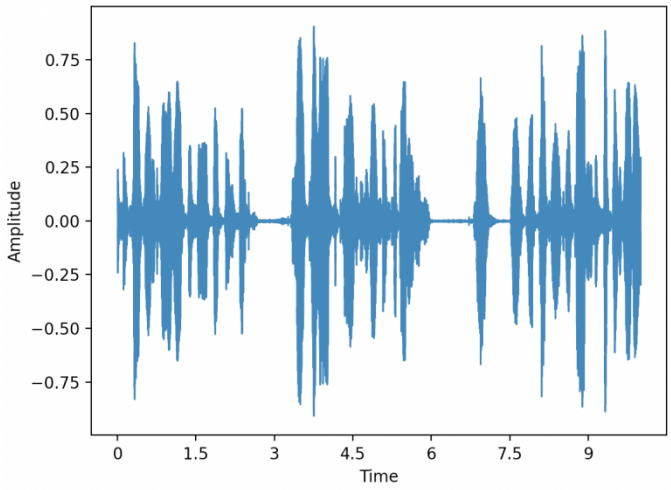
流利的音频波形图

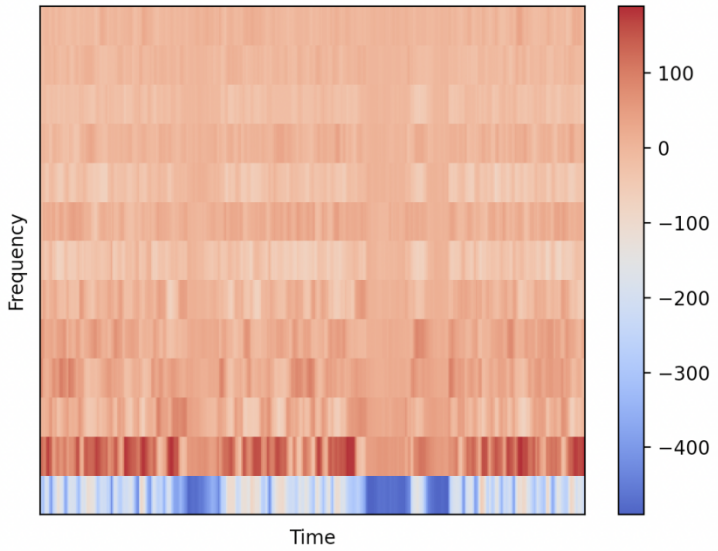

8192
4096
2048
1024
512
256
128
64
0
Frequency
0
1.5
3
4.5
6
7.5
9
Time
40
30
20
10
0
−10
−20
−30


8192
4096
2048
1024
512
256
128
64
0
Frequency
0
1.5
3
4.5
6
7.5
9
Time
40
30
20
10
0
−10
−20
−30


1000
Magnitude
Frequency
Time
200
100
0
−100
−200
−300
−400
−500


0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
Amplitude
0
1.5
3
4.5
6
7.5
9
Time


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Amplitude
0
1.5
3
4.5
6
7.5
9
Time



First, we used the Python pydub library to split the collected audio files of different formats into 10-second audio segments and saved them as MP3 files. We divided the collected audio files into a total of 447 10 -second audio segments, of which 232 segments were tagged as fluent audio and 215 segments were tagged as non-fluent audio.

Then we use the Python librosa library to iterate through each audio file and calculate the 25 Mel-spectral coefficients (MFCCs) using the following function .

```
signal, sample_rate = librosa.load(file_path, sr=22050)
mfcc = librosa.feature.mfcc(y=signal, sr=sample_rate, n_mfcc=25, n_fft=2048, hop_length=512)
```

25 commonly used MFCC coefficients, with a frame length of 2048 and a frame shift of 512. For a 10-second audio clip, the sampling rate was 22050 samples per second, generating 4 x 3 x 1 groups of 25 MFCC values in a two-dimensional matrix for each audio segment . We used the 2D matrices generated from 70% of the segments as the training set and the remaining segments as the test set for model training and testing.

Earlier, we mentioned labeling audio to distinguish between fluency and non-fluency. However, there isn't a single, universally accepted standard for defining fluency. Each language institution develops its own fluency scoring criteria based on internal parameters. In our study, our primary criterion was the presence of sound. Speech with numerous unnatural pauses or very slow delivery was typically labeled as non-fluent.

3. Experimental Framework

Our experimental procedure is based on standard deep learning methods. Our proposed method consists of three main steps: audio segmentation, feature extraction, and audio classification, ultimately classifying and predicting the most likely label for each individual audio segment. See Figure 1 for a detailed breakdown of the steps.

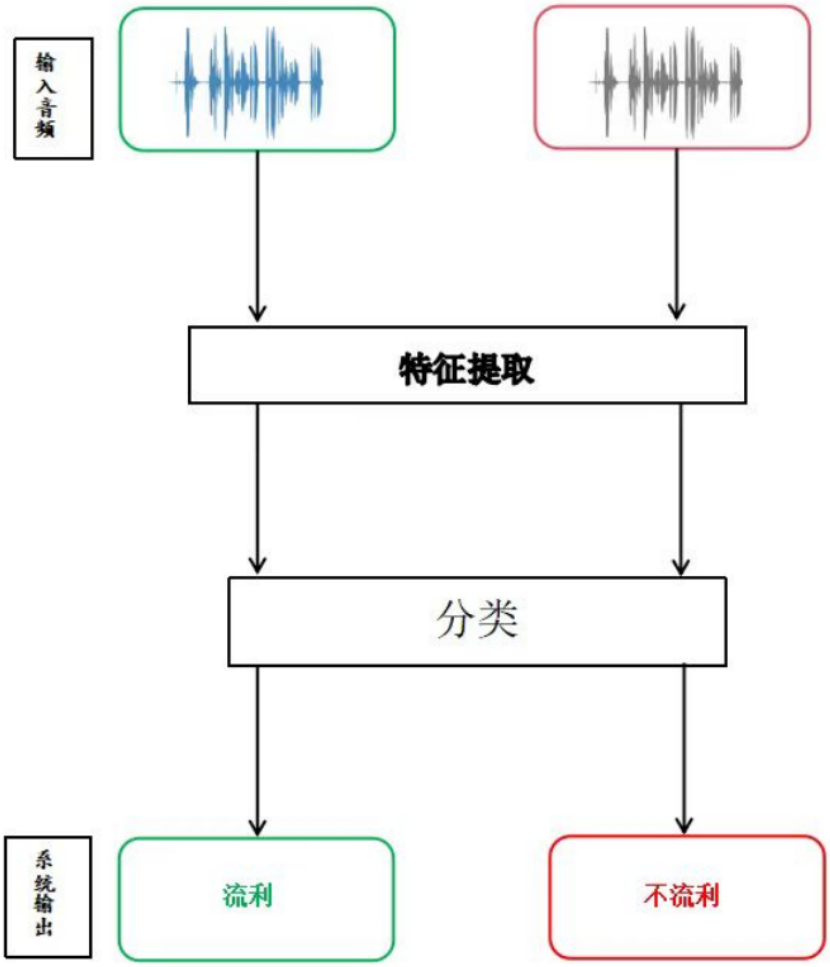


图 1 实验框架图

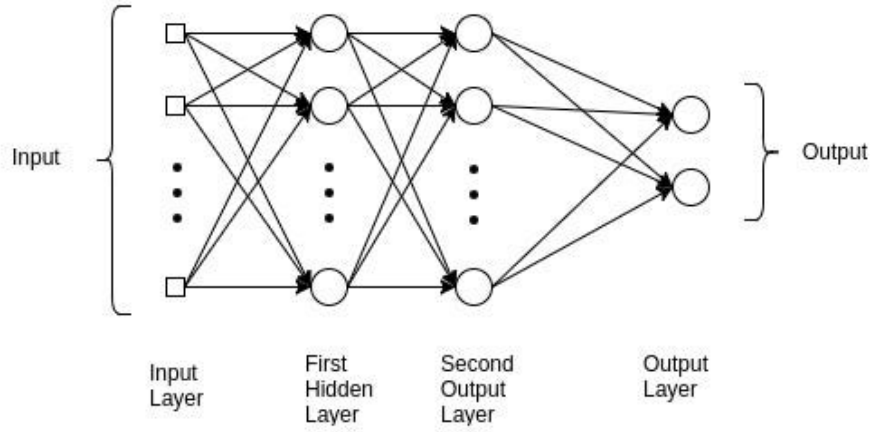
Audio segmentation is one of the most important preprocessing steps in most deep learning audio classification applications. Researchers frequently propose new segmentation frameworks to improve speech recognition applications. In our study, we segmented audio files into 10-second non-repeating audio segments. We used Python's pydub library to segment the audio files and saved the segmented audio files in MP3 format.

Feature extraction is a crucial step in deep learning. We wrote a Python script using the Python librosa library to extract features from audio segments. As mentioned earlier, we selected 25 MFCC coefficients as the feature values for each audio segment. Then, we stored the feature values of all 447 audio segments, along with their corresponding label values, into a single Jason file for quick data retrieval by the classification model below. See the appendix ` prepare_dataset.py` for the detailed script .

Audio classification is a crucial step in applying deep learning to assess English proficiency. In this work, we are interested in comparing different classification models, namely Multilayer Perceptron (MLP), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN). The main objective is to test the accuracy of common MFCC-based eigenvalue classification models. Due to the distinct image features in audio spectrograms reflecting fluency in spoken English— specifically, less fluent audio exhibits more and shorter graphical breaks—we also attempted to use state-of-the-art ResNet and ViT models in image recognition for classification , aiming for higher accuracy.

Multilayer perceptrons (MLPs) are a type of feedforward neural network that approximates complex functions through a multilayer network structure. They are trained using the backpropagation algorithm and the network weights are adjusted based on the gradient of the loss function. The output of each node in each layer introduces nonlinearity through the activation function, making the network simulation more powerful. A classic MLP consists of an input layer, an output layer, and one or more hidden layers. Each layer is fully connected and can approximate any continuous function. [4]

The schematic diagram is as follows:

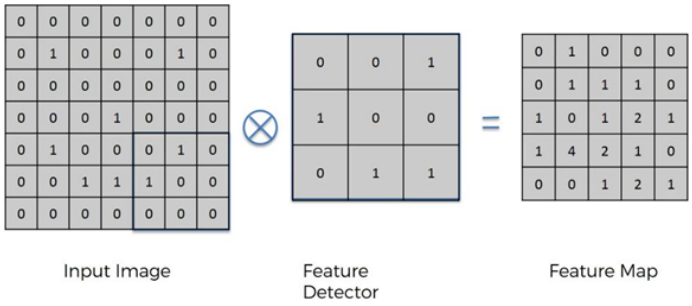


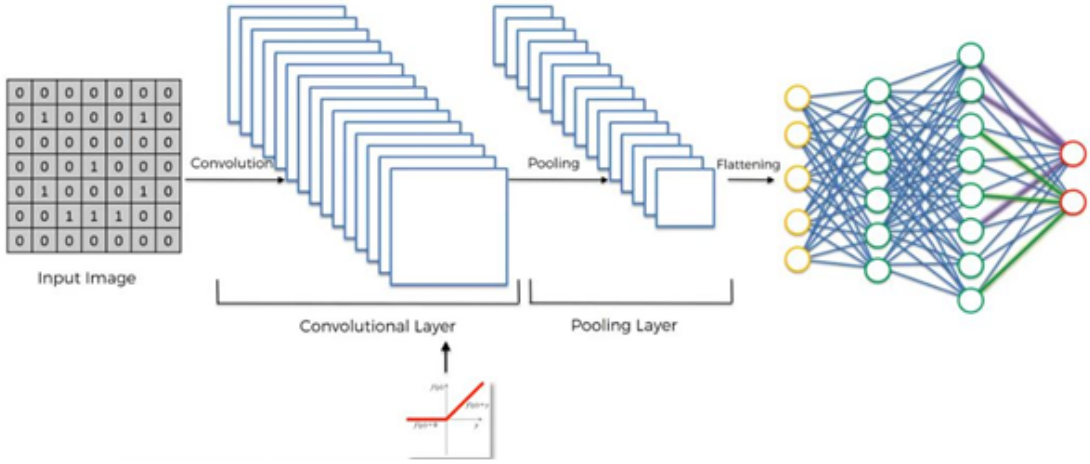
Our proposed multilayer perceptron architecture consists of three hidden layers with 512 × 256 × 64 neurons each. The final output layer comprises two neurons, one representing fluency and one representing non-fluency. Each neuron in the hidden layers uses the ReLU function as its activation function. The neurons in the output layer use a soft max function to convert the output into classification probabilities. Finally, the model predicts the label of the most probable class.

In terms of implementation, we used Keras, a submodule of Python Tensorflow . The script snippet below shows how to use Keras to build the multilayer perceptron model required for fluency recognition. The model input consists of 447 matrices of size 4 x 31 x 25. First, 70% (313 matrices) are randomly selected as the training set, and 134 are used as the test set. The first step involves flattening the training set matrices into a 1 x 07 x 75 one-dimensional array as the data input layer. This array is then passed sequentially to three hidden layers consisting of 512, 256, and 64 neurons respectively, before finally entering the output layer for data classification. We used the ReLU function as the activation function in the hidden layers and selected classification cross- entropy as the loss function combined with the softmax activation function in the output layer. To reduce overfitting , we opted for Dropout to discard 30% of the neurons in the hidden layers . We also used the advanced ADAM optimizer to improve classification accuracy . The detailed script can be found in the appendix `mlp.py` .

```
import tensorflow.keras as keras

model = keras.Sequential([
      # input layer
      keras.layers.Flatten(input_shape=(X.shape[1], X.shape[2])),
      # 1st dense layer
      keras.layers.Dense(512, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),
      keras.layers.Dropout(0.3),
     # 2nd dense layer
      keras.layers.Dense(256, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),
      keras.layers.Dropout(0.3),
      # 3rd dense layer
      keras.layers.Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),
      keras.layers.Dropout(0.3),
      # output layer
      keras.layers.Dense(2, activation='softmax')
  ])

  # compile model
  optimiser = keras.optimizers.Adam(learning_rate=0.0001)
  model.compile(optimizer=optimiser,
            loss='categorical_crossentropy',
```

CNNs use convolution operations to extract spatial information from input images and employ weight-sharing convolution kernels for feature extraction. After downsampling by pooling layers, the most salient features are pooled. Convolutional layers and pooling layers are stacked alternately to form a deep network that extracts multi-level features. CNNs have achieved great success in image processing due to their advantage in extracting spatial information through convolutional computation. [5]





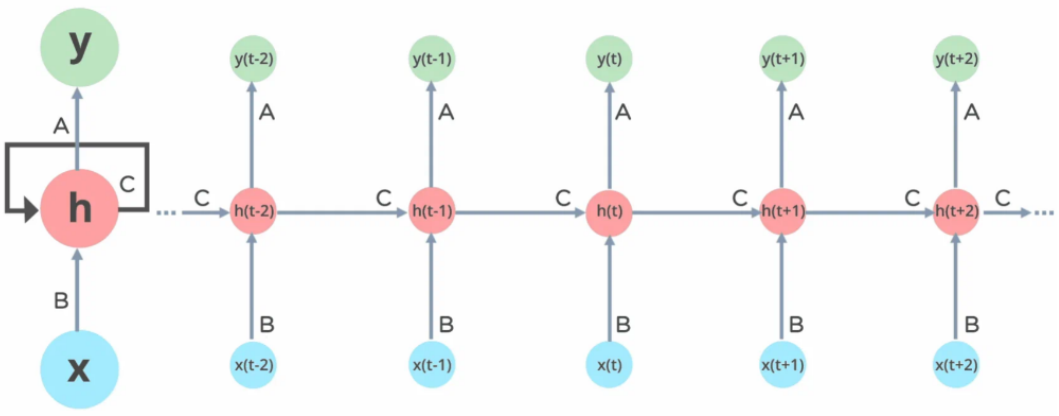
The convolutional neural network architecture used in our experiment has four hidden layers. The first two have 64 convolutional filters, the last has 32 convolutional filters, and the last is a fully connected layer with 32 neurons. The output layer consists of two neurons, which correspond to our two categories, similar to a multilayer perceptron. The activation function of the hidden layers is ReLU , while the activation function of the output layer is softmax .

The specific implementation is as follows. Similar to MLP, we used Keras, a submodule of Python Tensorflow . The model input consists of 447 matrices of size 443x25. First, 70% (313 matrices) are randomly selected as the training set, and 134 are used as the test set. The first step involves adding a depth of 1 dimension to the 4x31x25 two-dimensional array of the training set , forming a 4x31x25x1 three - dimensional array as the input layer, which is then passed to a convolutional layer with 64 filters and dimensions of 3 (width and height). The output is a 429x23x64 three - dimensional array . Then, it passes through a pooling layer with a window size of 3 (width and height) and a vertical and horizontal stride of 2,

reducing the output dimension to 215x1x2x64 . Two more convolutional pooling layers further reduce the output dimension to 53x2x32 . The data is then flattened into a one-dimensional array of 3392, which is passed to a hidden layer consisting of 32 neurons and a fully connected classifier consisting of 2 output units. Finally, the predicted probability values for the two fluency categories are calculated. See the appendix cnn.py for the detailed script .

```
import tensorflow.keras as keras

model = keras.Sequential()
    # 1st conv layer
    model.add(keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())
    # 2nd conv layer
    model.add(keras.layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())
    # 3rd conv layer
    model.add(keras.layers.Conv2D(32, (2, 2), activation='relu'))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())
    # flatten output and feed it into dense layer
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(32, activation='relu'))
    model.add(keras.layers.Dropout(0.3))

    # output layer
    model.add(keras.layers.Dense(2, activation='softmax'))
```

Recurrent Neural Networks (RNNs) are a type of neural network capable of processing sequential data. Their distinguishing feature is the directed loop formed between nodes in the network. This recurrent structure allows RNNs to retain their internal state, storing past information, and calculating the current output based on the current input and past states. RNNs can be viewed as networks with "memory." The core mechanism of RNNs is the introduction of this recurrent network structure. At time t, the hidden state h of the network is influenced not only by the current input x but also by the hidden state h_{t-1} from the previous time step. This loop embodies the "memory" mechanism.
Specifically, an RNN defines a recursive function f. At time t, h_t = f(x_t, h_{t-1}). The function f represents the transformation relationship between the current input x_t and the previous state h_{t-1}. This transformation method gives RNN the ability to process sequential data.



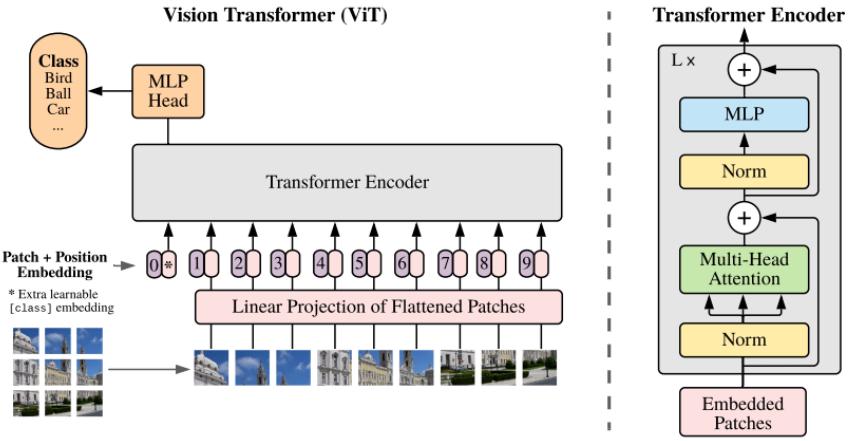
[6]

When dealing with long sequences of data, RNN units are prone to gradient vanishing, causing them to possess only short-term memory. This means that RNNs can only access information from more recent sequences and lack memory for earlier sequences, resulting in information loss. To address this issue, the

research community has proposed a Long Short-Term Memory (LSTM) structure as an improvement over general RNNs. In this study, we implemented a recurrent neural network using an LSTM structure . The specific implementation is as follows, using the Keras submodule of Python Tensorflow . The model input consists of 447 two-dimensional arrays of size 4 x 31 x 25. First, 70% (313 arrays) are randomly selected as the training set, and 134 arrays as the test set. The first step involves passing the 4 x 31 x 25 two-dimensional arrays from the training set to the first LSTM layer with an output dimension of 256. Then, it is passed to the second LSTM layer with an output dimension of 64, further reducing the output dimension to 64. The data is then passed to a hidden layer consisting of 64 neurons and a fully connected classifier consisting of 2 output units, ultimately calculating the predicted probability values for the two fluency categories. See the appendix rnn.py for the detailed script .

```
import tensorflow.keras as keras
model = keras.Sequential()

model.add(keras.layers.LSTM(256, input_shape=input_shape,
return_sequences=True))
model.add(keras.layers.LSTM(64))

model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(0.3))

# output layer
model.add(keras.layers.Dense(2, activation='softmax'))
```

ViT is a model proposed by the Google team in 2020 that applies Transformer to image classification. Although it is not the first paper to apply Transformer to vision tasks, it has become a milestone in the application of Transformer in the field of computer vision (CV) because its model is "simple" and effective, and has strong scalability (the larger the model, the better the effect). It has also sparked subsequent related research. Its principle is as follows [7] :



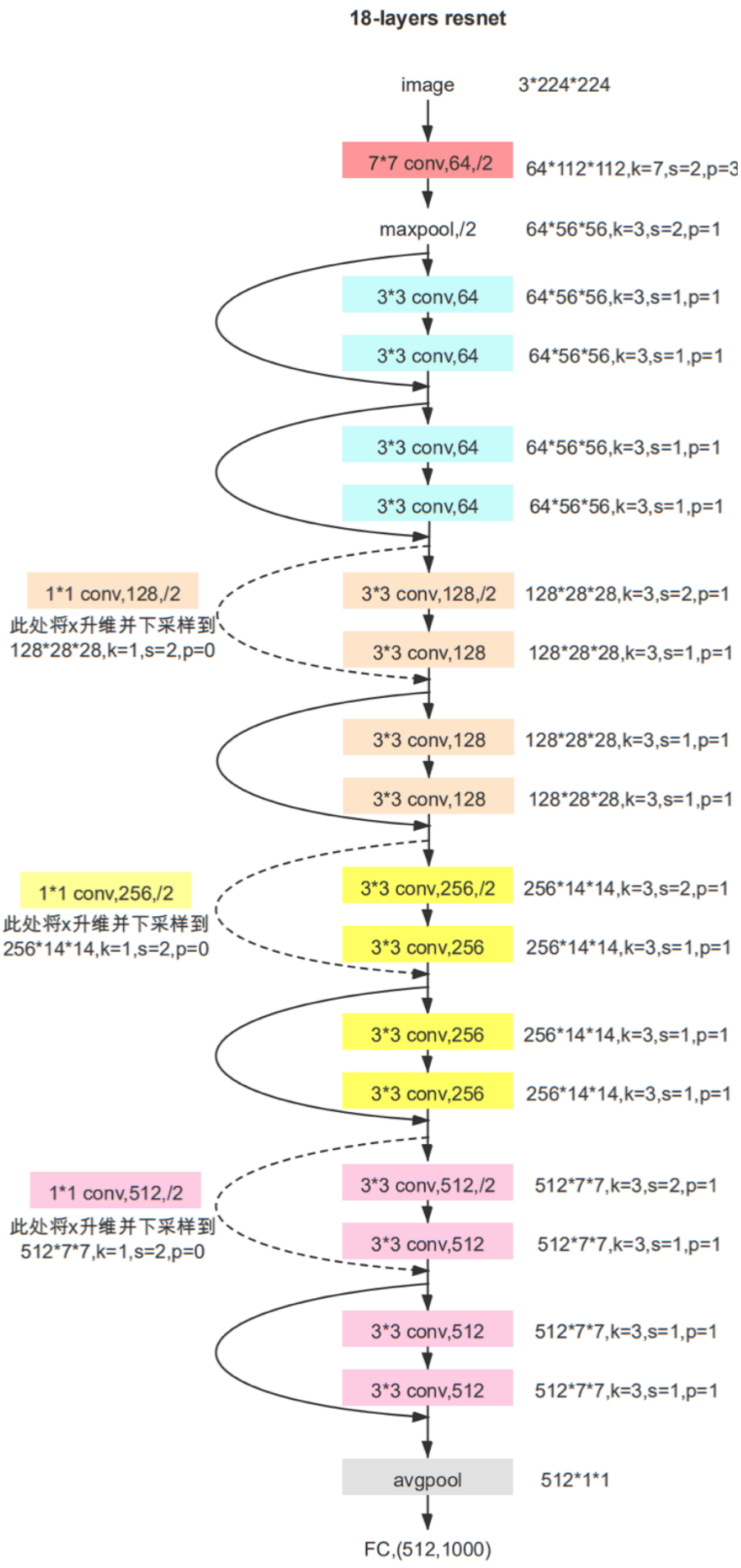
1. Input image: The image is segmented into smaller patches (16x16 pixels).
2. Patch embeddings (Patch + position Embeddings): Each image patch is mapped to a vector representing its content. Position embeddings are added to preserve spatial information.
3. Transformer Layers: Patch token sequences are fed into the standard transformer encoder architecture. Self-attention layers model the global relationships between all patches.
4. Classification Head: Output transformer features are passed to the MLP head for classification into image categories.
5. End-to-end training: The ViT model is trained end-to-end from image pixels to class labels, learning relevant visual features in a data-driven manner, which is different from CNN neural networks.

The core conclusion of the original ViT paper is that with sufficient pre-training data, ViT outperforms CNNs, overcoming the limitation of transformers lacking inductive bias and achieving better transfer learning results in downstream tasks . In this study of English fluency, we observed distinct graphical features in the audio spectrograms corresponding to different fluency levels : more fluent spectrograms have fewer obvious signal interruptions, while less fluent spectrograms have more. Following the ViT approach , we first extracted the audio spectrograms of the audio files as input features for the ViT model. Since the ViT model requires a large amount of training data, and our dataset was relatively small, using it to build the ViT model resulted in poor performance. We then attempted transfer learning, using a pre-trained ViT-16 model and default weights provided by PyTorch. The ViT-16 model used the large Google internal dataset (JFT-300M), which contains approximately 300 million labeled images. Its wide range of categories enables the Vi T-16 pre-trained model to achieve good performance in various image classification tasks.

The specific implementation is as follows. We still used the pre-trained ViT-16 model based on Python 's PyTorch . The model input consists of 447 audio spectrograms. First, 329 arrays were randomly selected as the training set, and 118 arrays as the test set. The first step was to adjust the spectrograms to 224x224 images with three primary colors. The images were then divided into 196 16x16 segments . Image patching , with the added patch token sequence, is fed into a standard transformer encoder. After 12 transformer encoder passes, a 1x768 data set is generated. This data is then passed to a fully connected classifier consisting of 768 input units and 2 output units, ultimately calculating the predicted probabilities for the two fluency categories. See the appendix `vit_with_pretrain.py` for the detailed script.

```
# 1. Get pretrained weights for ViT-Base
pretrained_vit_weights = torchvision.models.ViT_B_16_Weights.DEFAULT

# 2. Setup a ViT model instance with pretrained weights
pretrained_vit = torchvision.models.vit_b_16(weights=pretrained_vit_weights).to(device)

# 3. Freeze the base parameters
for parameter in pretrained_vit.parameters():
    parameter.requires_grad = False

# 4. Change the classifier head
class_names = ['bad','good']

set_seeds()
pretrained_vit.heads = nn.Linear(in_features=768, out_features=len(class_names)).to(device)
pretrained_vit # uncomment for model output

from torchinfo import summary
# Print a summary using torchinfo (uncomment for actual output)
summary(model=pretrained_vit,
        input_size=(32, 3, 224, 224), # (batch_size, color_channels, height, width)
        # col_names=["input_size"], # uncomment for smaller output
        col_names=["input_size", "output_size", "num_params", "trainable"],
        col_width=20,
        row_settings=["var_names"]
)
```

Residual Neural Network (ResNet) was proposed by Kaiming He , Xiangyu Zhang, and others from Microsoft Research and published at the prestigious 2016 Computer Vision Conference CVPR with the title "Deep Residual Learning for Image Recognition". [9] Residual Neural Network performs well in ImageNet image classification tasks and has been widely used in various tasks in the field of computer vision, such as image classification, object detection, and semantic segmentation. The main innovation of ResNet is the introduction of "residual blocks" or "skip connections", which solves the gradient vanishing problem common in deep network training by directly skipping inputs to subsequent layers, making the network efficient and stable . ResNet18 is the shallowest network architecture in the ResNet series, with a depth of 18 layers, specifically 17 convolutional layers plus 1 fully connected layer. Its network structure is as follows.


18-layers resnet
image
3*224*224
7*7 conv,64,/2
64*112*112,k=7,s=2,p=3
maxpool,/2
64*56*56,k=3,s=2,p=1
3*3 conv,64
64*56*56,k=3,s=1,p=1
3*3 conv,64
64*56*56,k=3,s=1,p=1
3*3 conv,64
64*56*56,k=3,s=1,p=1
3*3 conv,64
64*56*56,k=3,s=1,p=1
1*1 conv,128,/2
此处将x升维并下采样到
128*28*28,k=1,s=2,p=0
3*3 conv,128,/2
128*28*28,k=3,s=2,p=1
3*3 conv,128
128*28*28,k=3,s=1,p=1
3*3 conv,128
128*28*28,k=3,s=1,p=1
3*3 conv,128
128*28*28,k=3,s=1,p=1
1*1 conv,256,/2
此处将x升维并下采样到
256*14*14,k=1,s=2,p=0
3*3 conv,256,/2
256*14*14,k=3,s=2,p=1
3*3 conv,256
256*14*14,k=3,s=1,p=1
3*3 conv,256
256*14*14,k=3,s=1,p=1
3*3 conv,256
256*14*14,k=3,s=1,p=1
1*1 conv,512,/2
此处将x升维并下采样到
512*7*7,k=1,s=2,p=0
3*3 conv,512,/2
512*7*7,k=3,s=2,p=1
3*3 conv,512
512*7*7,k=3,s=1,p=1
3*3 conv,512
512*7*7,k=3,s=1,p=1
3*3 conv,512
512*7*7,k=3,s=1,p=1
avgpool
512*1*1
FC,(512,1000)

Input layer: Accepts an image of size 224x22 4x3 and passes it through a 7x7 convolutional layer with 64 filters, a stride of 2, and a large kernel to extract basic image features. Then, a 3x3 max pooling layer is applied to further reduce the image size.

Residual Blocks: This part is the main innovation of the ResNet network. ResNet18 has four stages , each containing two residual blocks. Each residual block contains two 3x3 convolutional layers with batch normalization and ReLU activation . The convolutional layers in each stage of the residual block have different numbers of filters: 64, 128, 256, and 512. The feature map size is halved in each stage, and the feature value output after four residual block stages is 512x7x7. Finally, global average pooling is used to further reduce the dimensionality to a fixed-size vector of 512x1x1.

Fully connected layer: The vector after global average pooling is input into the fully connected layer, and the output consists of two nodes, corresponding to the two categories of English fluency. See the appendix resnet.py for the detailed script.

In Table 1, we summarize the above architecture.

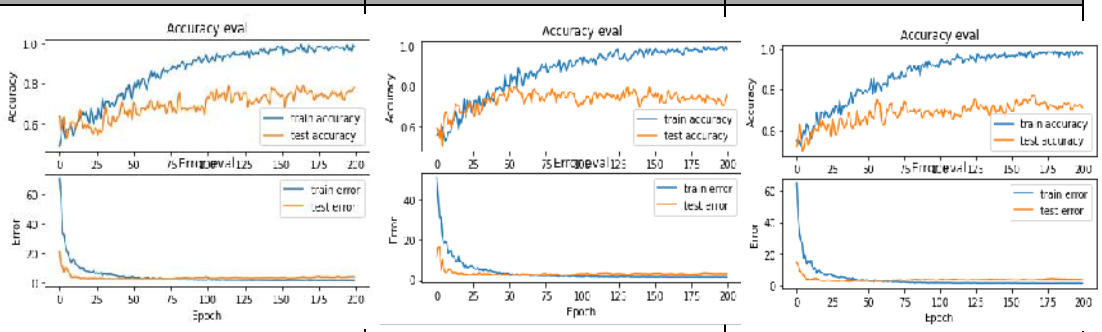
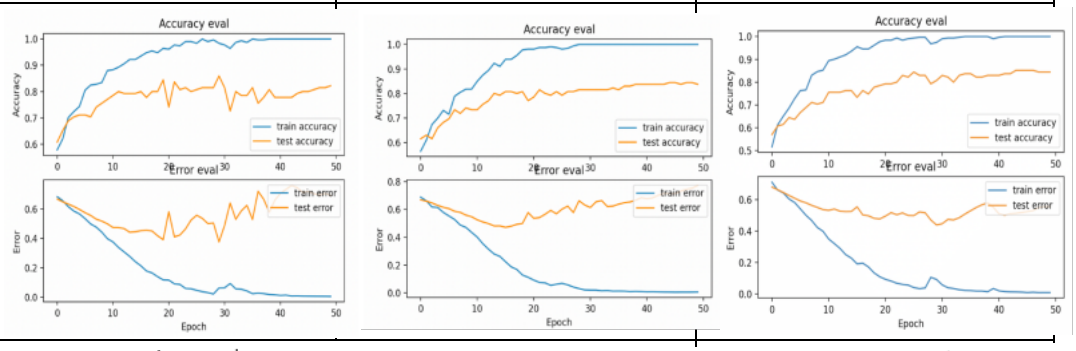
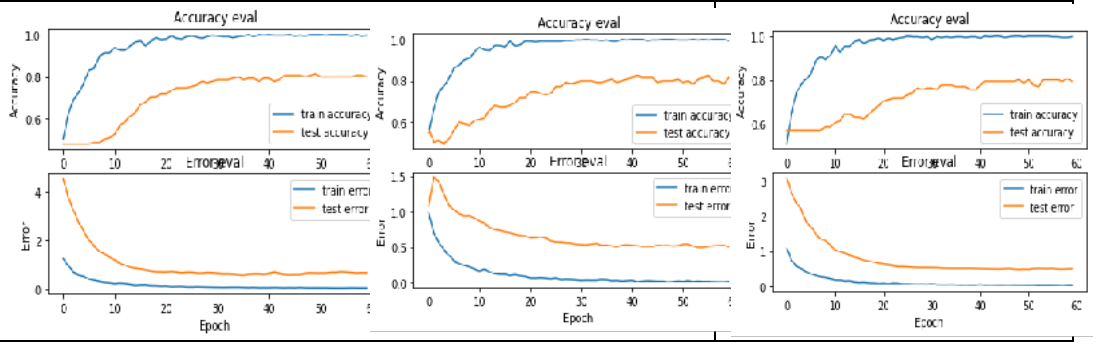
Table

Model type	Hidden layer	neurons	Activation function
Multilayer perceptron	3	512 × 256 × 64	Relu , softmax
Convolutional Neural Networks	4	64 × 64 × 32 × 32	ReLU, softmax
Recurrent Neural Networks	3	256 × 64 × 64	ReLU, softmax
Visual transducer (ViT)	12 (Transformer Encoder)	Embedded layer: 196x768 Transformer encoder layers (per layer): 196X768 + 196X3072 + 196X768	GELU, softmax
Residual Network (ResNet)	18	Input layer: 224x224x3 Residual block, first stage: 56x56x64x2 Second stage of residual block: 28X28X128X2 Residual block third stage: 14X14X256X2 Residual block fourth stage: 7X7X512X2	ReLU, sigmoid, softmax

4. Experimental Results

In this section, we will present our feature extraction and classification results. The main data analysis tools we used in our experiments were Anaconda (Python 3.8), commonly used Python data analysis packages such as Pandas, Keras (based on Tensorflow), Scikit-learn , and audio processing packages such as Librosa and Pydub.

We conducted three experiments each on three classification models based on MFCC feature values (MLP , CNN , and RNN) , and three experiments on large-scale models ResNet and ViT based on spectrograms as feature values . We plotted the changes in accuracy and error on the training and test sets in each training epoch and recorded the accuracy on the test set in the final training epoch . All five deep learning models we used were able to meet the fluency classification requirements. Their test set accuracy, from highest to lowest, was ViT > ResNet > RNN > CNN > MLP.

Model type	First experiment	Second experiment	Third experiment
Multilayer perceptron			
Recurrent Neural Networks			
Convolutional Neural Networks			

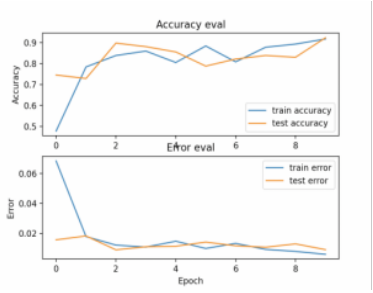
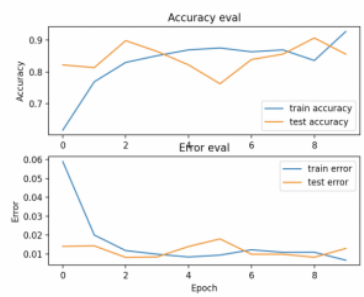
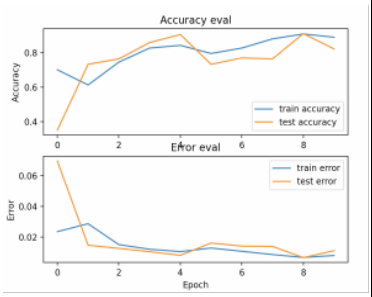
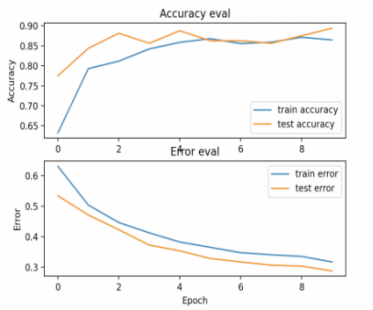
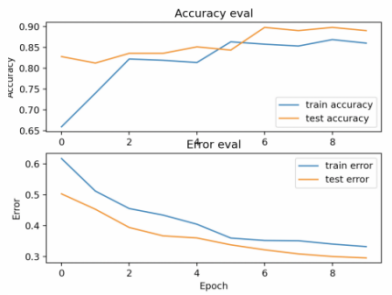
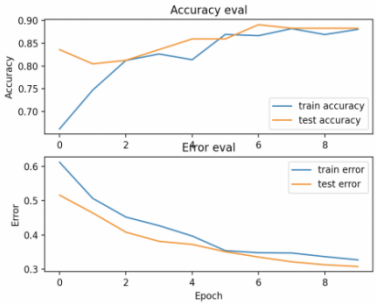
Model type	First experiment	Second experiment	Third experiment
Residual Networks (ResNet)			
Visual transducer (ViT)			

Table 2 Comparison of test accuracy of different deep learning models

Model type	First experiment	Second experiment	Third experiment
Multilayer Perceptron (MLP)	78.21%	79.26%	77.04%
Recurrent Neural Network (RNN)	82.22%	83.70%	84.44%

Convolutional Neural Network (CNN)	81.48%	82.22%	80.74%
Residual Networks (ResNet)	92.37%	85.59%	86.39%
Visual transducer (ViT)	89.06%	89.38%	88.28%

5. Conclusion

In this study , we proposed an audio processing method to determine the fluency of spoken English, extracting 447 non-overlapping audio segments. After audio segmentation and feature extraction, we tested five different neural networks as audio classification methods. In longitudinal comparisons, among MFCC-based training methods, RNN achieved a median accuracy of 83.7% on the test set , followed by CNN and MLP . Among spectrogram-based large language models, Vit generally outperformed the ResNet model. In our several experiments , we found that the Vit model , a state-of-the-art model in image recognition, achieved a median accuracy of 89.06 % , slightly better than the ResNet network's median result of 86.39 % .
Although we have achieved a relatively high accuracy rate, there is still room for improvement.

- Because our definition (labeling) of whether some audio is fluent or not during data collection is not very clear, we need to summarize it more clearly.
- In deep learning neural network architectures, we can still try to optimize the model by adding specific hidden layers and modifying the number of neurons in each layer , thereby achieving higher accuracy.
- The parameters of a pre-trained large language model may have been trained based on other types of images (such as landscapes and animals), and may not be suitable for spectrogram images.
- Considering that this work only defines two fluency levels, in more practical oral assessments, we need to define more fluency classification levels .

It's important to note that while ResNet and ViT models outperformed RNNs and CNNs in this study , it's inaccurate to conclude that pre-trained large language models are better suited for this task . The performance of each architecture depends on various factors, such as usage, data size, training time, parameter tuning, and hardware memory and computing power. Two methods based on different training sets coexist, organically integrating audio and image classification.

References

[1] Top 10 Strategic Technology Trends for 2024. Gartner, Inc.
https://www.gartner.com/en/newsroom/press-releases/2023-10-16-gartner-identifies-the-top-10-strategic-technology-trends-for-2024

[2] Speaker Fluency Level Classification Using Machine Learning Techniques. Alan Preciado-Grijalva, and Ramon F. Brena (2018)

[3] Paralinguistic and spectral feature extraction for speech emotion classification using machine learning techniques . Tong Liu & Xiaochen Yuan . EURASIP Journal on Audio, Speech, and Music Processing Volume 2023, Article number: 23 (2023)

[4] https://www.analyticsvidhya.com/blog/2020/12/mlp-multilayer-perceptron-simple-overview/

[5] ImageNet Classification with Deep Convolutional Neural Networks - Krizhevsky et al. (2012)

[6] https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

[7] An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Alexey DosoViTskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby (2020)

[8] Comparison of different implementations of MFCC. Feng Zheng, Guoliang Zhang and Zhanjiang Song published in the Journal of Computer Science and Technology (Vol. 16, No. 6, pp. 582-589) in 2001.

[9] Deep Residual Learning for Image Recognition. Kaiming He , Xiangyu Zhang , Shaoqing Ren , Jian Sun (2015)