

利用人工智能进行英语流利度分类

英语流利度的分类在英语学习中非常重要，但同时也需要老师花费大量精力去做出评价，所以我们可以用人工智能的方式自动评价学生的英语流利度就非常有用，能让学生得到口语练习的快速反馈。本文主要是通过几个常用的深度学习的模型，来评估英语口语的流利度。同时本文也尝试引入在图形识别领域比较先进的 ViT 模型，达到进一步提高模型准确性的目标。首先，我们需要建好研究需要的数据集。然后我们把搜集到的音频分割成 10 秒不重复的音频片段进行语音特征提取。我们从音频中提取常用的 25 个梅尔倒谱系数。在一共 447 个音频片段中，70% 为训练集，30% 为测试集。从这些片段中，我们选取的深度学习模型基本都能达到 80% 的测试准确度。ResNet 和 ViT 模型的测试准确率接近 90%。

1. 介绍

人工智能自上世纪五十年代开始提出后，经历了两次高潮和低谷。近十年来，随着计算力的瓶颈被打破，同时数字化时代的到来，给深度学习提供更多的数据，人工智能进入了飞速发展的时期，尤其是 2022 年 ChatGPT 横空出世以后，迅速成为全球关注的热点。2023 年 10 月 Gartner 发布的全球前十战略技术潮流 2024，5 个是和人工智能相关的。^[1]

英语作为三大主要科目，几乎所有的中国学生都投入大量时间学习英语，为了提高英语作为和国际人士沟通的有效性，学习英语过程中，我们越来越重视口语水平的提高。但我们想探索使用人工智能技术，快速区分英语片段的流利度。

流利度事实上是没有一个统一的定义。事实上，每个语言机构都根据其内部参数建立了一个评分的流利度指标。在我们的案例中，为了评估说话者的流利程度，我们讨论的流利度的指标主要基于声音及说话没有不自然的停顿。如果说话缓慢或停顿较多，那么这会影响说话者的流利度的分类。同时，流利和熟练是有区别的。这意味着流利是指能够感到舒适、听起来自然，并且能够随意操纵句子的所有部分的人。^[2]

这次的研究主要是通过使用深度学习模型，基于音频分析来区分英语口语的流利度。首先需收集英语口语音频，将这些音频文件根据其实际流利度打上低流利度和高流利度的标签。具体音频数据集的收集在第二章详细介绍。然后我将搜集到的音频分割成 10 秒不重复的音频片段，对其进行特征提取。然后提取音频的特征向量，使用深度学习的模型对音频的口语流利度进行分类。我把搜集到的音频分割成 10 秒不重复的音频片段，对其进行特征值的提取。具体特征值包括梅尔倒谱系数（MFCC）及音频的声谱图。随后，将数据集的特征值输入到分类模型中进行训练并使用测试准确率指标评估其性能。我们比较了三种常用的深度学习模型，即多层感知器（MLP）、卷积神经网络（CNN）、递归神经网络（RNN）根据 MFCC 的特征值进行分类。同时我们也尝试在图形识别领域比较先进的 ViT 模型和 ResNet 模型根据声谱图进行分类，以达到较高的准确性。对于常见深度学习模型，我们选取了 25 个 MFCC 梅尔倒谱系数

（MFCC）作为音频文件的特征值。基于流利度的高低在其音频声谱图上有较为明显的特征，即流利度较低的音频频谱图有比较多的断续区域，而流利度高的音频频谱图较少断续区域，我们也尝试使用在图像识别领域应用较多且准确度较高的 ViT 模型和 ResNet 模型进行流利度的分类。因为该模型主要基于图像识别，我们采用声谱图作为音频文件的特征值。

我们最后的分类实验数据表明，对于英语流利度的分类，常用的三个深度学习模型来说，卷积神经网络优于多层感知器和递归神经网络。多层感知器近 75%，卷积神经网络 86% 及递归神经网络 79% 的测试准确率。但基于声谱图识别的 ResNet 和 ViT 模型可以达到接近 90% 的测试准确率，略优于基于 MFCC 常用的三个深度学习模型。

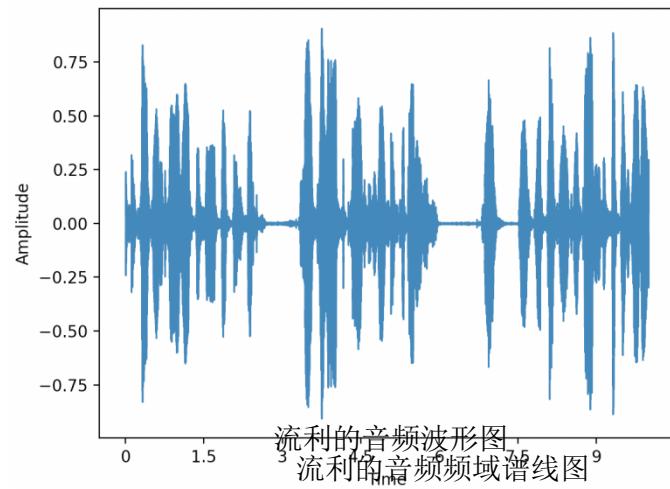
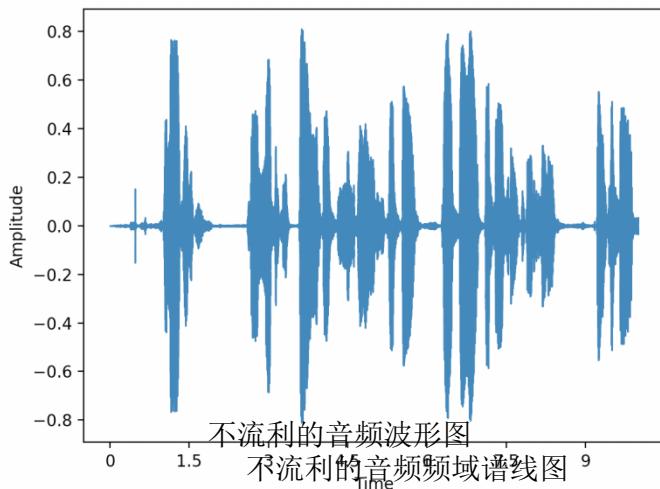
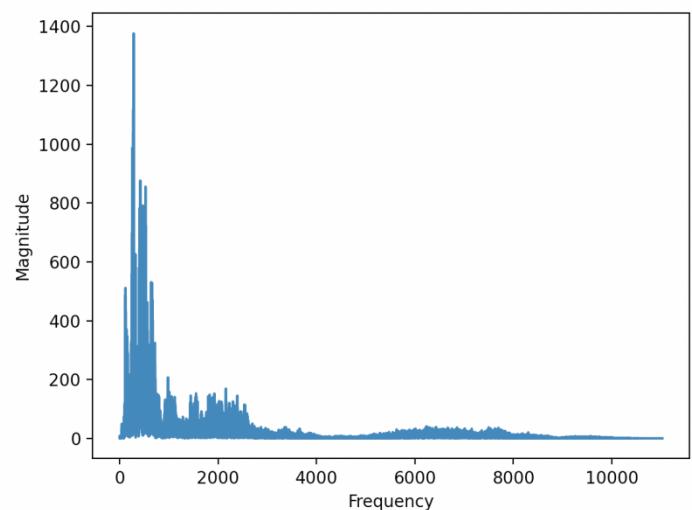
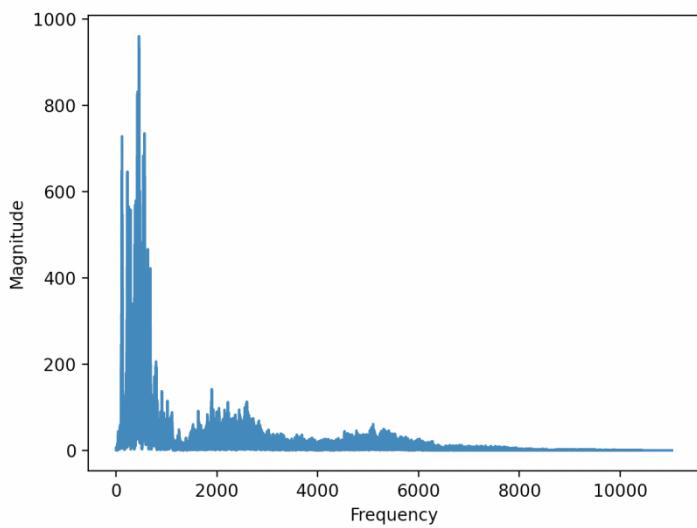
2. 准备数据集

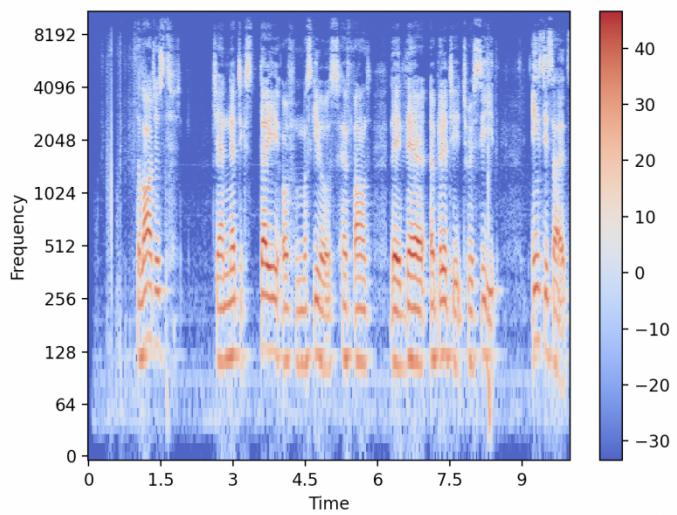
高质量的数据集是很多深度学习项目的必要条件。深度学习模型结构的主要难点是能够掌握数据之间的模式，从而使其足够复杂，能够执行准确的预测或者分类。如果数据集中包含错误标记或丢失的数据，那模型的准确性会受到影响。另一方面，模型的技术短板也与可使用的计算能力和训练所需的标记数据量有关。不幸的是，目前没有公开的数据集来满足此研究目的。我只能搜集了做此研究需要的数据集，

找同学有意识的流利阅读和不流利阅读来收集数据集。其中包括来自 10 个男生和 5 个女生时常约为 1 小时左右的不同流利度（高，低）的带标签的英语朗读音频。音频分割是大多数音频应用中最重要的预处理步骤之一。我们把搜集到的音频分割成音频片段，对其进行特征提取。在我们的研究中，主要采用的分割方法是将音频文件切割成 10 秒的非重叠片段。音频片段的切割主要使用名为 pydub 的 Python 库。

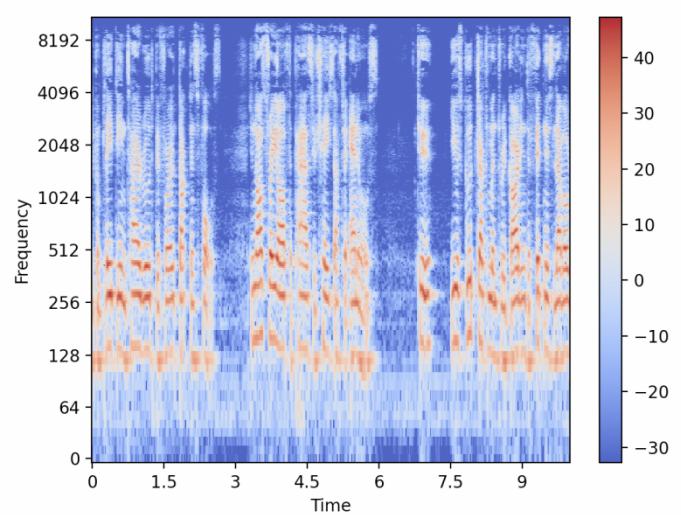
对于音频片段的特征值，我们选择从音频中提取梅尔倒谱系数，梅尔倒谱系数(MFCC: *Mel Frequency Cepstral Coefficient*) 是一种广泛用于自动语音和说话者分类的功能，因为它们适合于理解人类和人类说话的频率。MFCC 特征提取包括两个关键步骤：梅尔频率分析和倒谱分析。MFCC 是用于建立梅尔倒谱的一组关键系数。从音频信号中的片段，我们可以获得一组倒谱，这些倒谱足以表示音频信号。与一般的倒频谱不同，梅尔倒频谱上的频带在梅尔标度上均匀分布，这样的频带将比我们通常看到的更线性。倒谱表示方法更接近于人类的非线性听觉系统。因此，在该研究中，我们选择梅尔倒谱系数作为特征提取方法[3]。根据以往的研究，25 个梅尔倒谱系数能在音频分类领域提供更好的结果，故我们选择 25 个梅尔倒谱系数作为常用的三个深度学习模型的输入数据。[8]

我们编写了一个 python 脚本，使用名为 Librosa 的 python 包为创建的音频片段执行特征提取（在许多功能中，Librosa 通常用于特征提取，允许计算 30 多个音频特征）。音频帧 f_0, \dots, f_n 因此具有它们相应的特征向量 p_0, \dots, p_n 。Librosa 还可以使用 `display.specshow` 功能绘制音频频谱图。为了完整起见，我们展示了单个音频片段的以下常见频谱图。

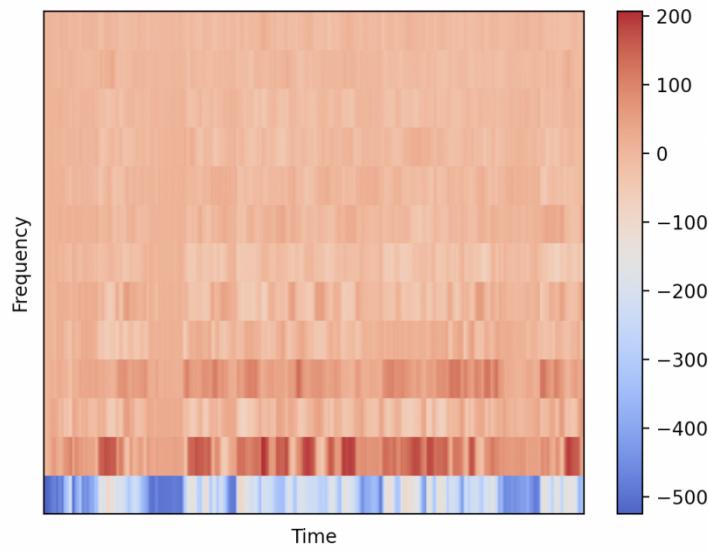




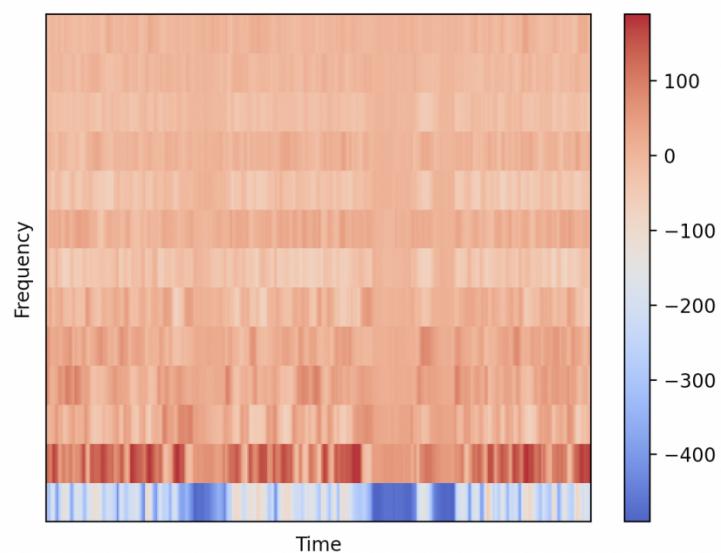
不流利的音频声谱图



流利的音频声谱图



不流利的音频梅尔倒谱系数图



流利的音频梅尔倒谱系数图

首先我们把收集到的不同格式语音文件，通过 Python 的 pydub 库，分割成 10 秒一段的音频文件，并保存为 mp3 格式。我们收集的音频文件一共切割成 447 个时长为 10 秒的音频片段，其中 232 个片段标签为流利的音频，215 个片段标签为不流利的音频。

然后我们使用 Python librosa 库遍历每个音频文件，通过以下的函数计算 25 个梅尔倒谱系数（MFCC）。

```
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)
```

我们选取比较常用的 25 个 MFCC 系数，选择帧长为 2048，帧移为 512. 对于一个 10 秒钟的音频，采样率为 22050 个/每秒，每段音频就会长生 431 组，每组 25 个 MFCC 值的二维矩阵。我们将 70% 的片段生成的二维矩阵设定为训练集，其余片段生成的为测试集，供模型训练和测试使用。

前面提到给音频打标签区分流利和不流利。其实没有一个统一标准来定义流利度。每个语言机构根据自己的内部参数建立了流利度的评分标准。在我们的研究中，我们的衡量标准主要是具有声音。如果说话中有较多不自然的停顿或非常缓慢，通常被打上不流利的标签。

3. 实验框架

我们的实验步骤是基于标准的深度学习的方法。我们的建议的方法包括主要三个步骤：音频分割，特征提取，音频分类等，最终归类和预测输出单个音频片段最有可能的标签。具体步骤可参考图 1.

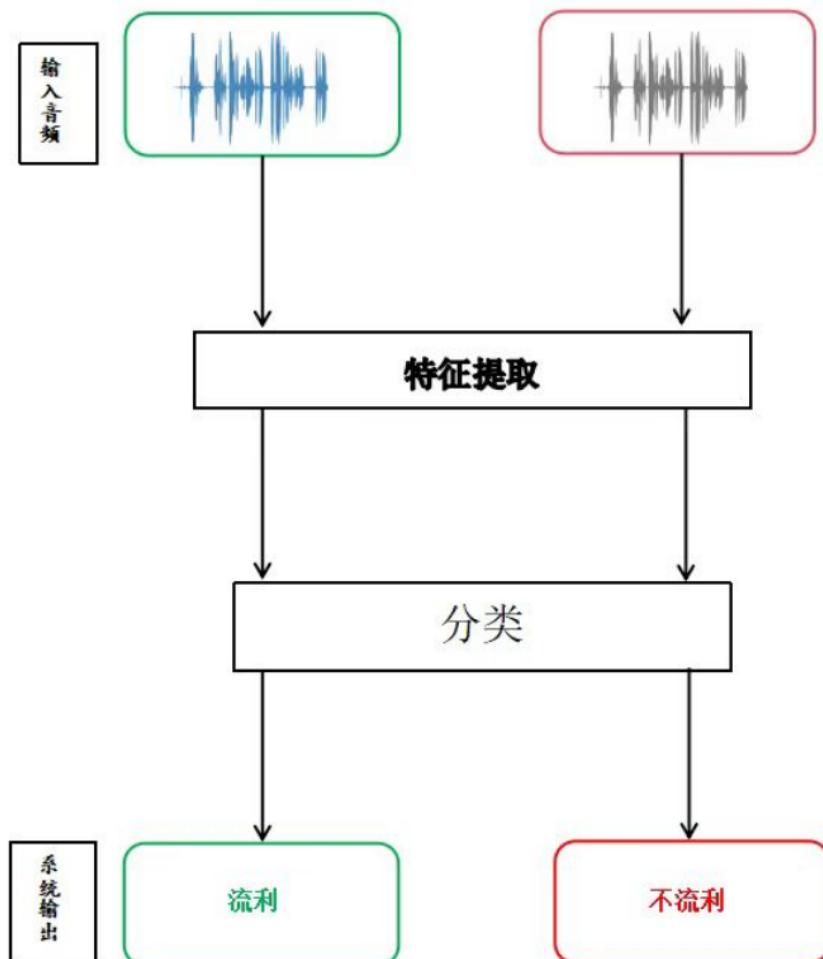


图 1 实验框架图

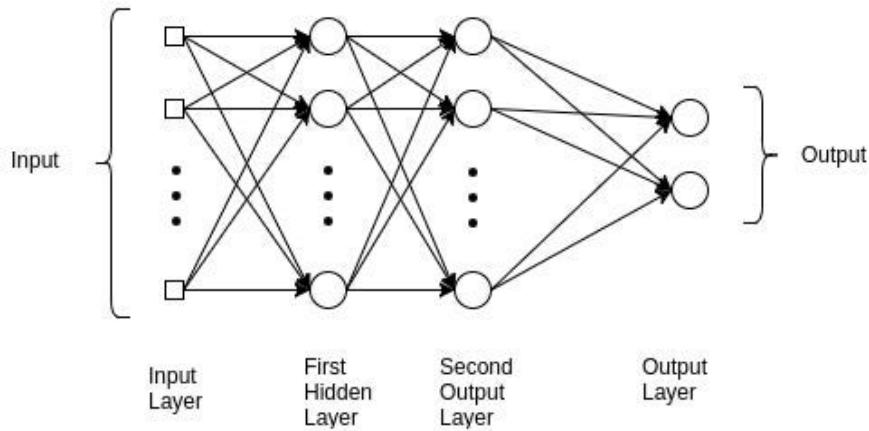
音频分割是大部分音频分类相关的深度学习最重要的预处理步骤之一。研究人员经常提出新的分割框架，以改进语音识别相关的应用。在我们的研究中，我们将音频文件分割成 10 秒不重复的音频片段，我们使用 Python 的 pydub 库分割音频文件，并将切割好的音频统一保存为 mp3 格式。

特征值提取是深度学习重要的环节，我们写了 Python 脚本，使用 Python librosa 库对音频片段进行特征值提取。如前所述，我们选取 25 个 MFCC 系数作为音频片段的特征值，然后将所有 447 个音频片段的特征值连同其对应标签值，存入一个 Jason 文件，便于下面的分类模型快速获取数据。具体脚本详见附录 prepare_dataset.py。

音频分类是应用深度学习判断英语熟练度的关键步骤。在这项工作中，我们感兴趣的是比较不同的分类模型，即多层感知器(MLP)、卷积神经网络(CNN)、递归神经网络(RNN)等。主要目标是在测试常见基于 MFCC 特征值分类模型的准确性。由于英语口语的流利度，在音频声谱图上有较为明显的图像特征，即不流利的音频有比较多且短的图形中断，我们也尝试使用在图形识别领域比较先进的 ResNet 和 ViT 模型进行分类，以期得到更高的准确率。

多层感知器(MLP)是一类前馈神经网络，通过多层网络结构逼近复杂函数。其利用反向传播算法训练，基于损失函数的梯度调节网络权重。每层节点输出通过激活函数引入非线性，使网络模拟算力更强。经典的 MLP 由输入层、输出层及一个或多个隐含层组成，每层完全连接，可以近似任意连续函数。[4]

原理示意图如下



我们尝试的多层感知器的架构包括 3 个 $512 \times 256 \times 64$ 神经元的隐藏层。最后的输出层包括 2 个神经元，1 个代表流利，1 个代表不流利。每个在隐藏层的神经元使用 relu 函数作为激活函数。输出层的神经元使用 softmax 函数将输出转成分类概率。最后，模型预测的标签为可能性最高的分类。

在具体程序实施上，我们采用了 Python Tensorflow 的子模块 Keras。下面的脚本片段包括了如何使用 Keras 模块构建流利度识别所需的多层感知器模型。模型的输入为 447 个 431×25 的矩阵。首先随机选取 70% 即 313 个矩阵为训练集，134 个为测试集。第一步训练集的矩阵压扁为一个 10775 的一维数组作为数据输入层，然后依次传递给 3 个分别由 512, 256 及 64 个神经元组成的隐藏层，最后进入输出层，提供数据分类。我们在隐藏层使用 relu 函数作为激活函数，在输出层选择了分类交叉熵作为损失函数搭配 softmax 激活函数。为了降低过拟合(Overfitting)的影响，我们选择对隐藏层的神经元 Dropout 的方式丢弃 30% 的神经元减少过拟合的现象。同时我们也选取了比较先进的 ADAM 优化器，提高分类的准确率。具体的脚本详见附录 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',
              metrics=['accuracy'])

#train model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size=64,
epoches=200)

#plot accuracy and error as a function of the epochs
plot_history(history)

```

CNN 利用卷积操作提取输入图像的空间信息,采用权值共享的卷积核进行特征提取。经过池化层的下采样后汇聚最显著特征。卷积层和池化层交替堆叠构成深层网络提取多级特征。CNN 通过卷积计算提取空间信息的优势,使其在图像处理方面有巨大的成功。[5]

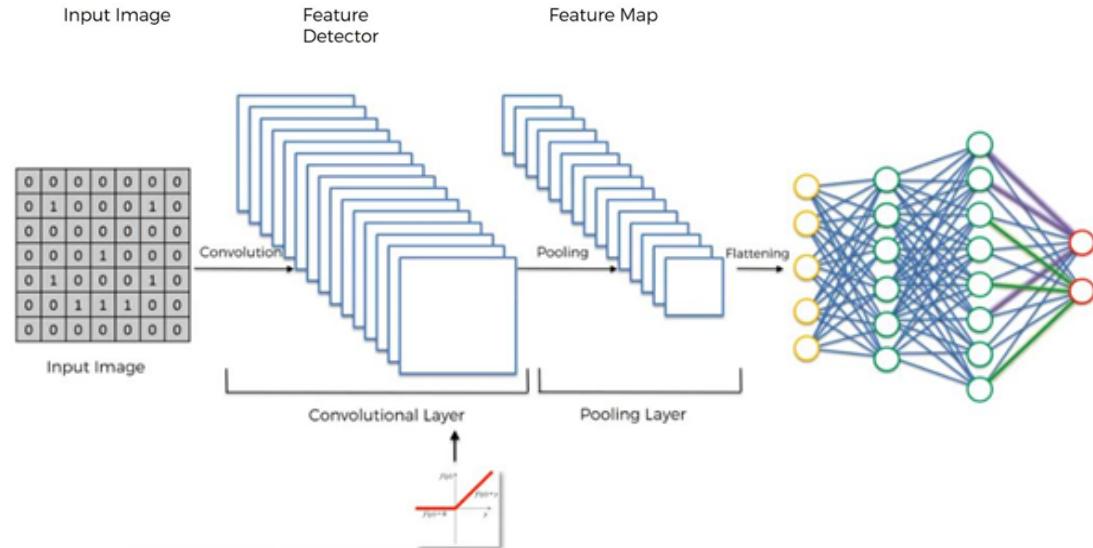
0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

 \otimes

0	0	1
1	0	0
0	1	1

=>

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1



我们实验使用的卷积神经网络架构有四个隐藏层，前两个有 64 个卷积滤波器，后一个有 32 个卷积滤波器，最后一个为 32 个神经元的全连接层。输出层由 2 个神经元组成，这 2 个神经元对应于我们的 2 个类别，类似于多层感知器，隐藏层的激活函数均为 relu，而输出层是 softmax 函数。

具体的程序实施如下，和 MLP 一样，我们采用了 Python Tensorflow 的子模块 Keras。模型的输入为 447 个 443X25 的矩阵。首先随机选取 70% 即 313 个矩阵为训练集，134 个为测试集。第一步训练集的 431X25 的二维数组，添加一个深度为 1 的维度，形成一个 431X25X1 的三维数组作为输入层，传递给一个有 64 个过滤器，宽高分别为 3 的卷积层。输出为 429X23X64 三维数组。然后经过一个池化窗口高宽分别为 3，垂直和水平步长均为 2 的池化层，降低了输出维度到 215X12X64。再经过两个卷积池化层，将输出维度进一步降低到 53X2X32。接着将数据压扁成一个 3392 的一维数组，传递给一个由 32 个神经元组成的隐藏层和一个由 2 个输出单元组成的全连接分类器，最终完成对应两个流利度分类的预测概率值的计算。具体脚本详见附录 `cnn.py`。

```

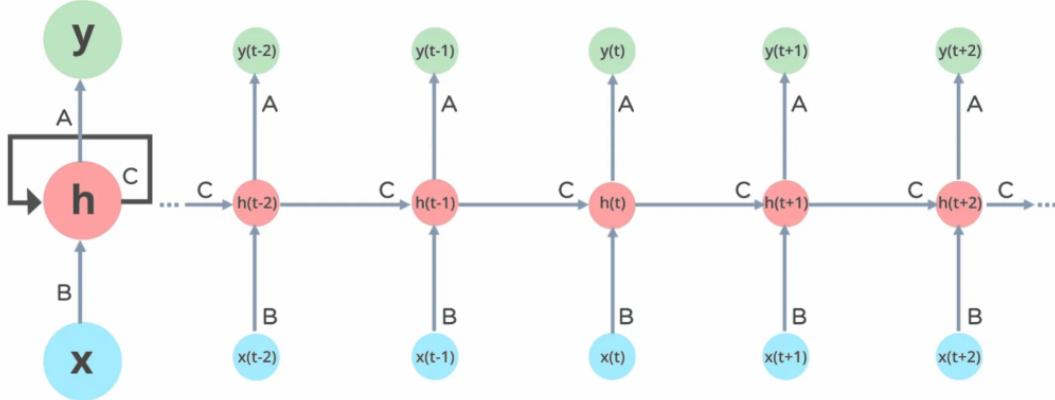
import tensorflow.keras as keras

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'))

```

RNN 是一类能够处理序列数据的神经网络。其区别在于网络中的节点之间形成一个有向循环。这种循环结构使得 RNN 可以保存过去信息的内在状态，并根据当前输入和过去状态进行当前输出的计算。可以把 RNN 看作有“记忆力”的网络。RNN 的核心机制是引入了循环的网络结构。在时刻 t , 网络的隐层状态 h 不仅受当前输入 x 的影响，也受到上一个时刻隐层状态 $h_{\{t-1\}}$ 的影响。这构成的循环体现了“记忆”机制。具体来说，RNN 中定义了一个递归函数 f 。在时刻 t , $h_t = f(x_t, h_{\{t-1\}})$ 。函数 f 代表当前输入 x_t 与前一状态 $h_{\{t-1\}}$ 之间的转换关系。这种转换方式赋予了 RNN 处理序列数据的能力。



[6]

RNN 单元在面对长序列数据时，很容易便遭遇梯度弥散，使得 RNN 只具备短期记忆，即 RNN 面对长序列数据，仅可获取较近的序列的信息，而对较早期的序列不具备记忆功能，从而丢失信息。为此，为解决该类问题，科研界提出了一种长短时记忆（LSTM）结构作为对于一般 RNN 的改良。在本次研究中，我们使用 LSTM 结构实施循环神经网络。具体的程序实施如下，我们还是采用了 Python Tensorflow 的子模块 Keras。模型的输入为 447 个 431X25 的二维数组。首先随机选取 70% 即 313 个数组为训练集，134 个为测试集。第一步训练集的 431X25 的二维数组，传递给第一个输出维度为 256 的 LSTM 层。然后传递给第二个输出维度为 64 的 LSTM 层，将输出维度进一步降低到 64。接着将数据传递给一个由 64 个神经元组成的隐藏层和一个由 2 个输出单元组成的全连接分类器，最终完成对应两个流利度分类的预测概率值的计算。具体脚本详见附录 rnn.py。

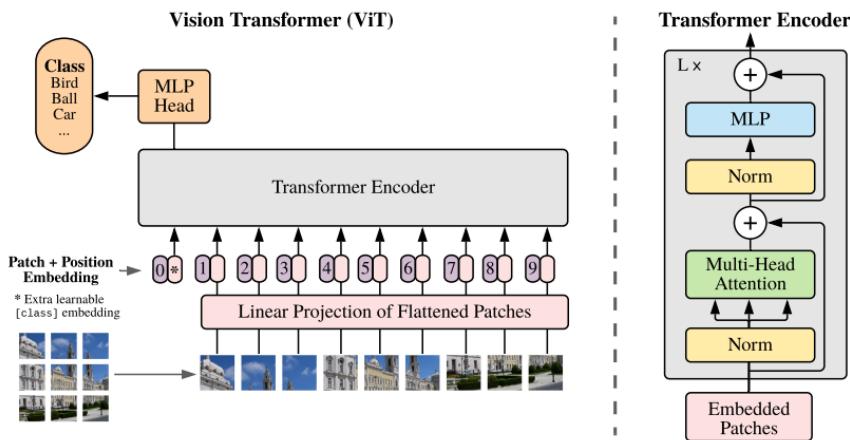
```
import tensorflow.keras as keras
model = keras.Sequential()

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 是 2020 年 Google 团队提出的将 Transformer 应用在图像分类的模型，虽然不是第一篇将 transformer 应用在视觉任务的论文，但是因为其模型“简单”且效果好，可扩展性强（scalable，模型越大效果越好），成为了 transformer 在计算机视觉(CV)领域应用的里程碑著作，也引爆了后续相关研究，其原理如下[7]：



1. 输入图像：图像被分割成更小的补丁（16x16 像素）。
2. 补丁嵌入(Patch+position Embeddings)：每个图像补丁都映射到表示内容的向量。添加位置嵌入以保留空间信息。
3. 转换器层(Transformer Layers)：补丁令牌序列被输入到标准变压器编码器架构中。自注意力层对所有补丁之间的全局关系进行建模。
4. 分类头(Classification Head)：输出变压器特征被传递到 MLP 头以分类为图像类别。
5. 端到端训练：ViT 模型从图像像素到类标签进行端到端训练，以数据驱动的方式学习相关视觉特征，这与 CNN 神经网络不同。

ViT 原论文中最核心的结论是，当拥有足够多的数据进行预训练的时候，ViT 的表现就会超过 CNN，突破 transformer 缺少归纳偏置的限制，可以在下游任务中获得较好的迁移效果。对于本次英语流利度的研究，我们可以在不同流利度对应的音频声谱图中发现较为明显的图形特征，即比较流利的声谱图较少明显的信号中断，不流利的声谱图有较多信号中断。根据 ViT 方案，我们先提取了音频文件的音频声谱图作为 ViT 模型的输入特征值。由于 ViT 模型需要大量的训练数据，我们本次研究搜集的数据集数量比较小，用我们的数据集构建 ViT 模型，性能不好。我们尝试迁移学习，使用了预训练的 ViT-16 模型和 PyTorch 提供的默认权重。ViT-16 模型使用的数据集为谷歌内部使用的大型数据集（JFT-300M），JFT-300M 数据集的规模非常大，包含约 3 亿张带标签的图像。其涵盖丰富多样的类别，使得 ViT-16 预训练模型在各种图像分类任务中取得良好的性能。

具体的程序实施如下，我们还是采用了预训练的基于 Python 的 Pytorch 的 ViT-16 模型。模型的输入为 447 个音频声谱图。首先随机选取 329 个数组为训练集，118 个为测试集。第一步将声谱图调整成 224X224 的 3 原色的图像。图像被分割成 196 个 16X16 的 图像补丁，加入补丁令牌序列输入给标准的变压器编码器。经过 12 次变压器编码器，最终生成 1X768 的数据，数据传递一个由 768 个输入单元和 2 个输出单元组成的全连接分类器，最终完成对应两个流利度分类的预测概率值的计算。具体脚本详见附录 vit_with_pretrain.py

```
# 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

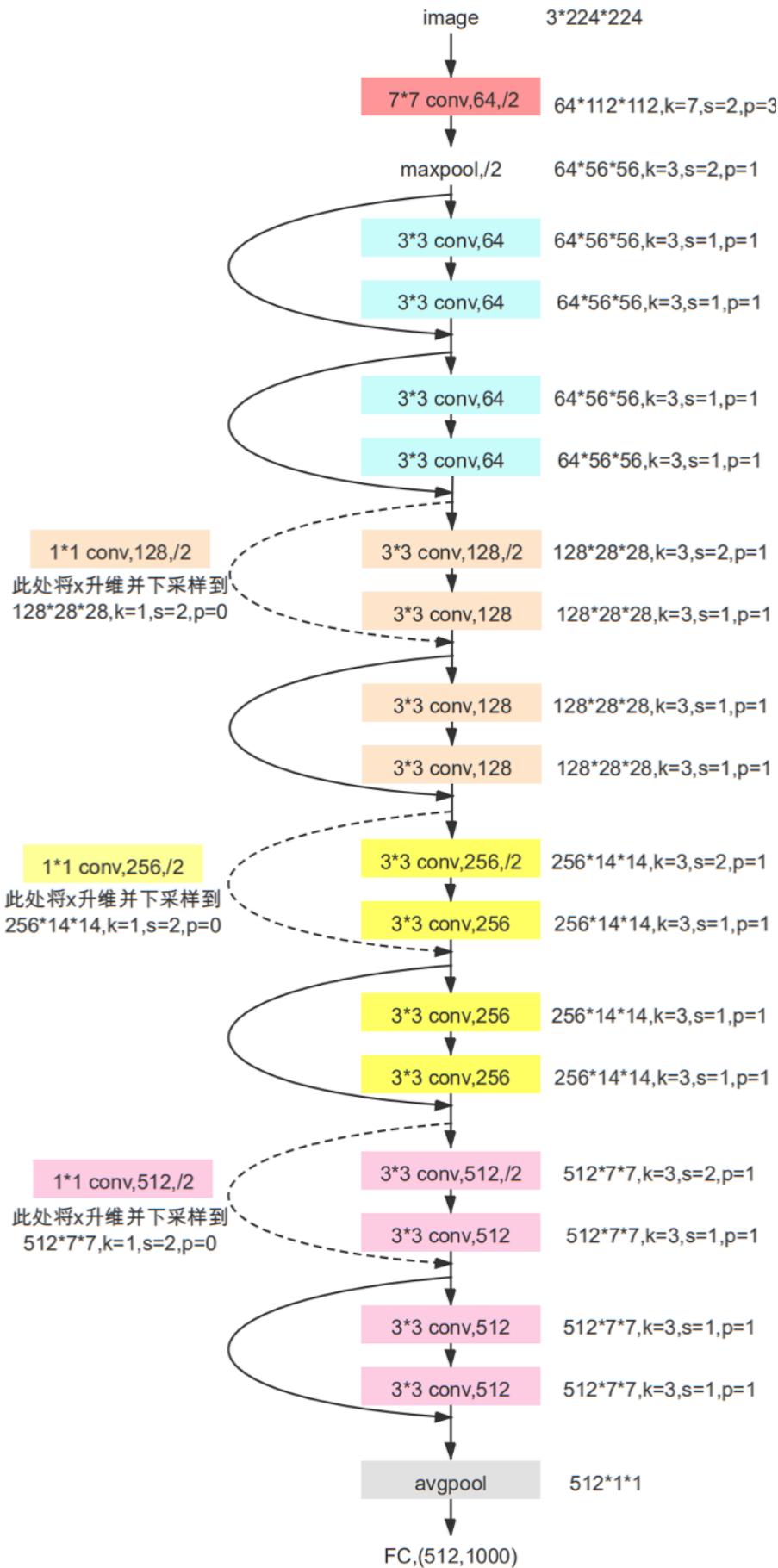
# 4. Change the classifier head
class_names = ['bad','good']

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
```

残差神经网络(ResNet)是由微软研究院的何恺明、张祥雨等提出，发表在著名的2016年计算机视觉会议CVPR上，论文标题为“Deep Residual Learning for Image Recognition”。[9] 残差神经网络在ImageNet图像分类任务中表现出色，被广泛应用于计算机视觉领域的各种任务，如图像分类、目标检测和语义分割。ResNet的主要创新在于引入了“残差块”或“跳跃连接”，通过将输入直接跳跃传递到后面的层，解决了深度网络训练过程中常见的梯度消失问题，使得网络可以高效和稳定。ResNet18是ResNet系列中最浅的一种网络架构，具有18层的深度，具体是17层的卷积层加上1层全连接层。其网络结构如下。

18-layers resnet



输入层： 接受大小为 224X224X3 的图像，并经过一个 7X7 的卷积层，使用 64 个滤波器，步长为 2，卷积核较大，用来提取基本的图像特征，然后接着一个 3X3 的最大池化层，进一步降低图像的尺寸。

残差块： 该部分为 ResNet 网络的主要创新，ResNet18 共有四个阶段(Stages)，每个阶段包含两个残差块。每个残差块包含两个 3x3 卷积层，具有批量归一化(Batch Normalization)和 ReLU 激活功能。每个阶段残差块的卷积层有不同数量的滤波器，分别为 64、128、256 和 512。特征图尺寸在每个阶段中减半，经过四个残差块阶段输出的特征值为 512X7X7。最后通过全局平均池化，进一步降维到一个 512X1X1 的固定大小的向量。

全连接层： 将经过全局平均池化的向量输入到全连接层，输出 2 个节点，对应英语流利度的两个分类。具体脚本详见附录 resnet.py

在表 1 中，我们总结了上述体系结构。

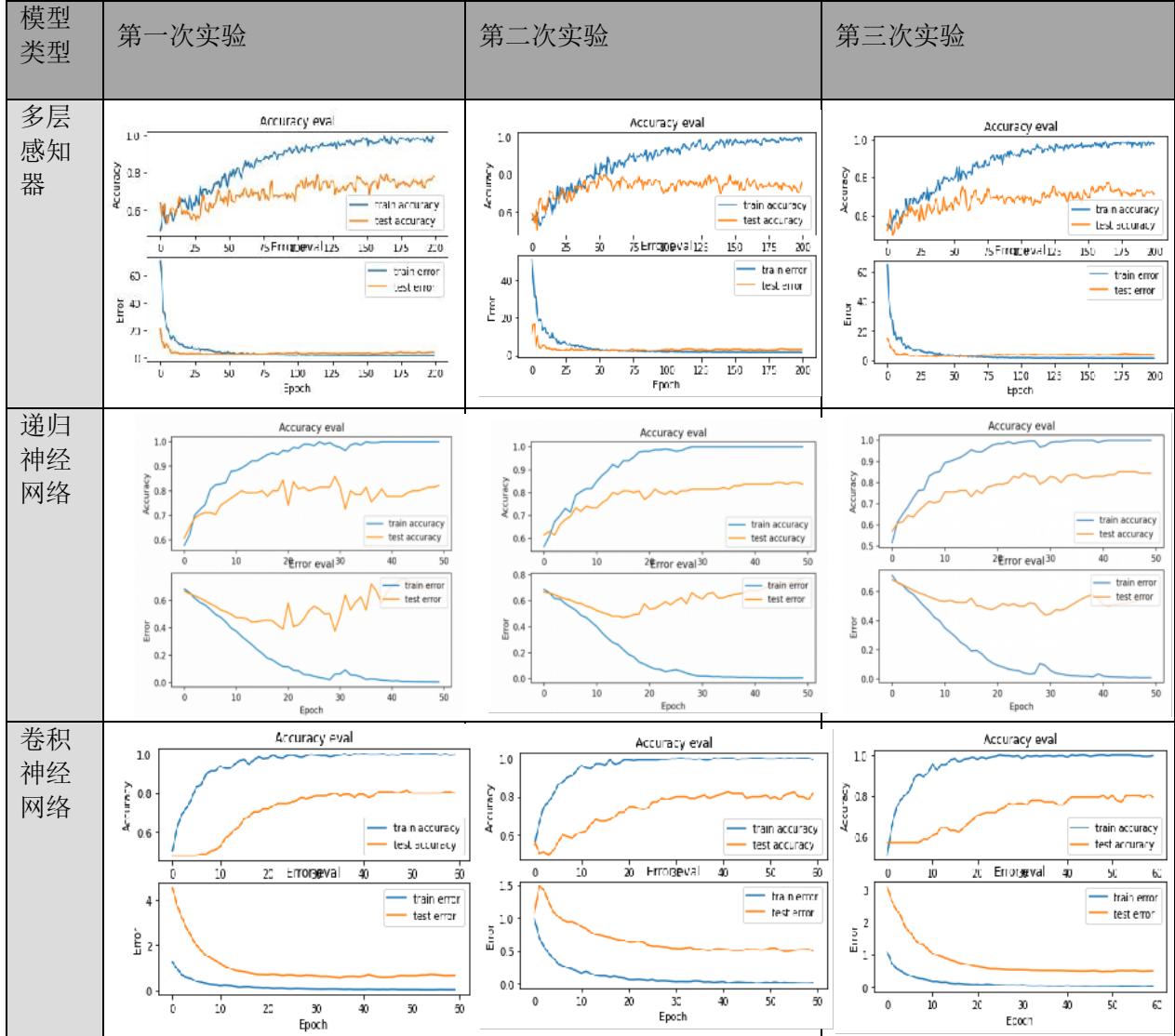
表 1 不同深度学习模型的基本结构信息

模型类型	隐藏层	神经元	激活函数
多层感知器	3	$512 \times 256 \times 64$	Relu, softmax
卷积神经网络	4	$64 \times 64 \times 32 \times 32$	Relu, softmax
递归神经网络	3	$256 \times 64 \times 64$	Relu, softmax
视觉转换器(ViT)	12 (Transformer 编码器)	嵌入层: 196×768 Transformer 编码器层(每层): $196 \times 768 + 196 \times 3072 + 196 \times 768$	GELU, softmax
残差网络 ResNet	18	输入层: $224 \times 224 \times 3$ 残差块第一阶段: $56 \times 56 \times 64 \times 2$ 残差块第二阶段: $28 \times 28 \times 128 \times 2$ 残差块第三阶段: $14 \times 14 \times 256 \times 2$ 残差块第四阶段: $7 \times 7 \times 512 \times 2$	Relu, sigmod,softmax

4. 实验结果

在本节中，我们将介绍我们的特征提取和分类结果。我们在实验中使用的主要数据分析工具是 Anaconda (Python 3.8)，常用的 Python 数据分析包 Pandas，Keras (基于 Tensorflow)、Scikit-learn、及音频处理包 Librosa、Pydub 等。

我们对三种基于 MFCC 特征值的分类模型(MLP, CNN 及 RNN)分别进行了 3 次实验，对基于声谱图为特征值的大模型 ResNet 和 ViT 也进行了三次实验，绘制训练集及测试集准确率及误差在每个训练轮次的变化图像并记录了测试集在最后一次训练轮次的准确率。我们采用的 5 个深度学习模型均能完成流利度分类的需要，其测试集准确度从高往低，依次为 Vit > ResNet >RNN>CNN>MLP.



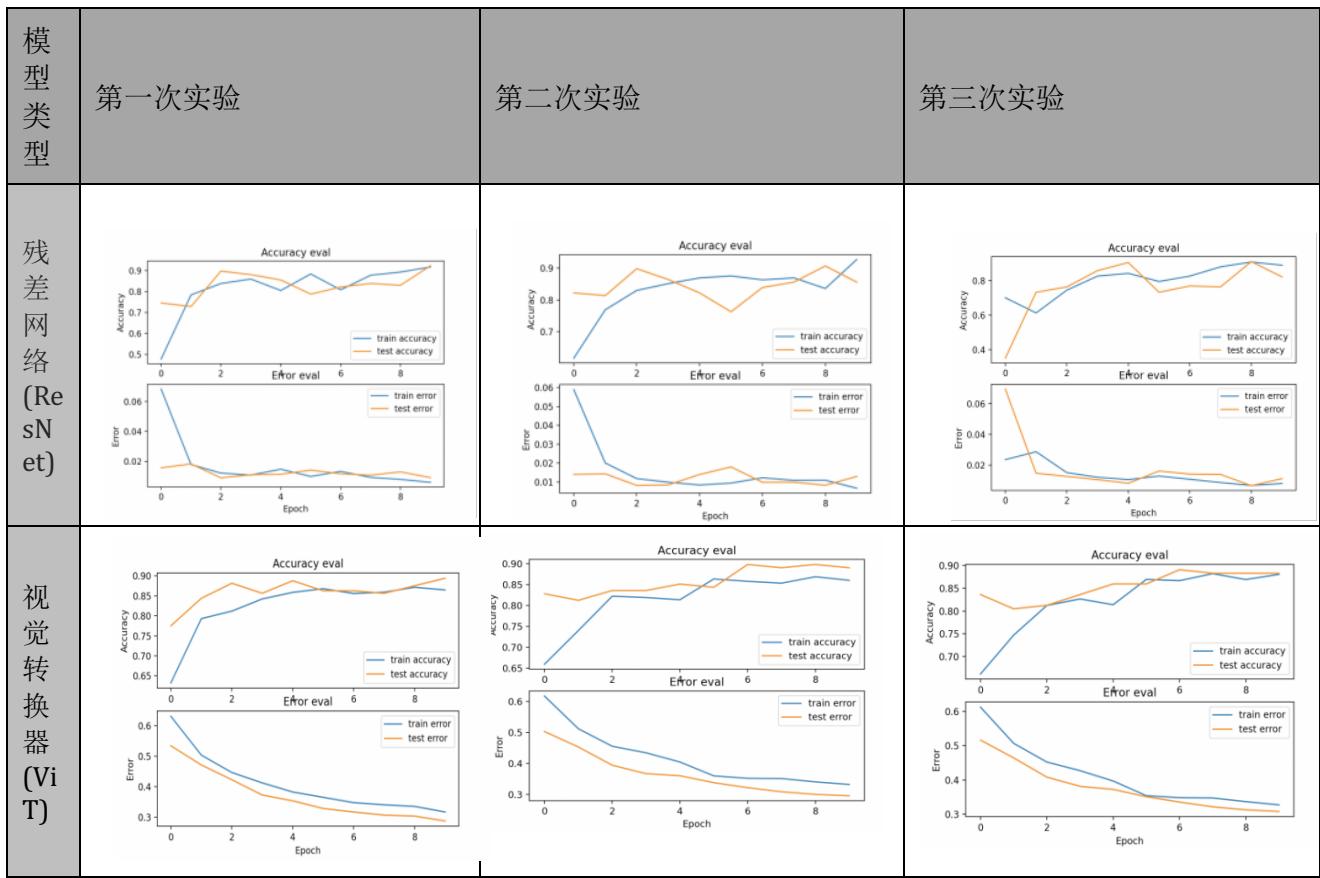


表 2 不同深度学习模型的测试准确率比较

模型类型	第一次实验	第二次实验	第三次实验
多层感知器 MLP	78.21%	79.26%	77.04%
递归神经网络 RNN	82.22%	83.70%	84.44%
卷积神经网络 CNN	81.48%	82.22%	80.74%
残差网络 (ResNet)	92.37%	85.59%	86.39%
视觉转换器 (ViT)	89.06%	89.38%	88.28%

5 结论

在这次的研究中，我们提出了一种能够确定英语口语流利程度的音频处理的方法，提取了 447 个不重叠的音频片段。经过音频分割和特征值提取等处理后，我们尝试了 5 种不同的神经网络作为音频分类的方

法。纵向对比，在基于 MFCC 的训练方法中，RNN 测试集准确率中位数为 83.7%、其次是 CNN, MLP; 在基于声谱图的大语言模型的方法中，ViT 的准确率普遍比 ResNet 模型高。在我们的数次实验中发现，图形识别领域比较先进的 ViT 模型的准确度中位数为 89.06%，略优于 ResNet 网络的中位数结果 86.39%。虽然我们所取得的准确率比较高，但仍有改进的空间。

- 由于我们在数据采集时对于部分音频流利与否的界定（做标签）不是非常清晰，因此需要更加清晰的归纳。
- 在深度学习神经网络架构下，我们仍然可以尝试添加特定的隐藏层并修改每层神经元的数量的方式来优化模型，从而达到更高的准确率。
- 预训练大语言模型已有的参数可能是根据其他类别的图片（如风景、动物）训练，无法贴合声谱图这一种类型的图片。
- 考虑到本次工作只定义了两个流利度级别，在更加实际的口语测评中，我们需要定义更多的流利度分类级别。

需要注意的是，ResNet 和 ViT 模型在本次研究中优于 RNN 及 CNN，但不能一概而论地认为预训练大语言模型更适合这次的任务。每个架构的性能取决于各种因素，如使用情况、数据规模、训练时间、参数调整、硬件的内存和计算能力等。两种基于不同训练集的方法是并存的，只是将音频分类与图像分类作有机融合。

参考资料

- [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 于 2001 年发表在《Journal of Computer Science and Technology》（第 16 卷，第 6 期，第 582-589 页）
- [9] Deep Residual Learning for Image Recognition. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015)