

# 哈尔滨工业大学

# 实验报告

## 实验（一）

题    目 语音识别实验

专    业 人工智能

学    号 2021110683

班    级 2103602

学    生 徐柯炎

指 导 教 师 郑铁然

实 验 地 点 格物 213

实 验 日 期 2024.4.14

计算机科学与技术学院

## 一、实验目的

### 1. 连续语音识别任务

- 1) 找到一种连续语音识别模型；
- 2) 进行算法复现；
- 3) 进行识别测试。

本次实验选择了任务二：连续语音识别任务，并采用 wav2vec2 进行实验。

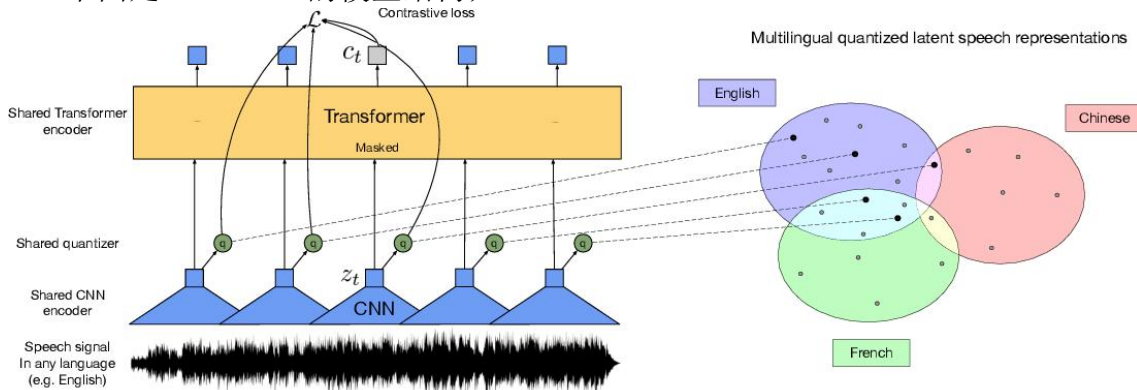
## 二、实验背景

本次实验选择了任务二：连续语音识别任务，并采用 wav2vec2 进行实验。

### 1. Wav2vec2 介绍

Wav2Vec2 是由 Hugging Face 和 Facebook 合作开发的一种音频特征提取模型，它是基于 Transformer 架构的自监督学习模型，用于处理音频数据。Wav2Vec2 是 Wav2Vec 模型的升级版，在许多语音处理任务中取得了更好的性能。

下面是 wav2vec2 的模型结构：



Wav2Vec2 模型有以下特点：

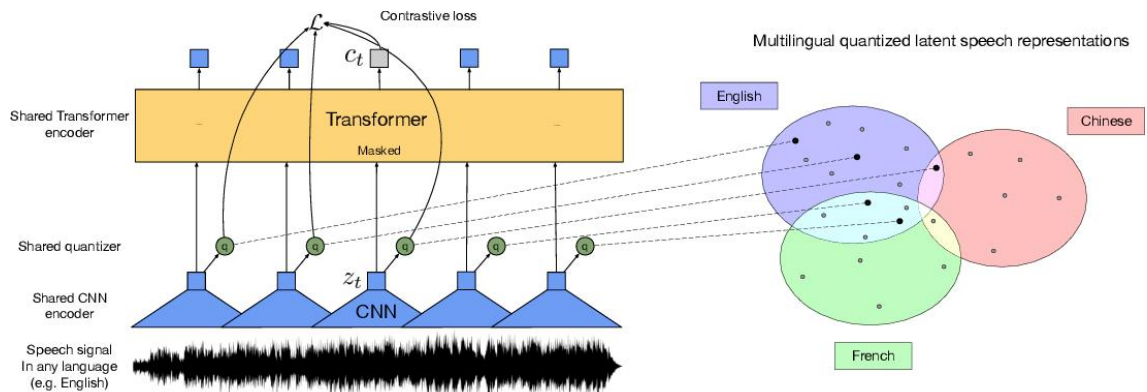
- 1) 自监督学习：Wav2Vec2 是通过自监督学习的方式进行训练的，它利用大规模的无标签音频数据来学习音频特征，无需人工标注数据。
- 2) 预训练任务：在预训练阶段，Wav2Vec2 通过掩码语言建模（Masked Language Modeling, MLM）和声学对比损失（Contrastive Acoustic Loss）等任务来学习音频特征。
- 3) 特征提取器：Wav2Vec2 使用了一种称为"Contextual Aggregator"的特征提取器，它能够捕捉音频片段的上下文信息，并生成高质量的音频特征表示。
- 4) Fine-tuning：预训练完成后，Wav2Vec2 可以用于各种语音处理任务的微调，比如语音识别、语音合成、情感识别等。微调过程通常会根据具体任务调整模型的最后几层或添加任务特定的头部。
- 5) 多语言支持：Wav2Vec2 支持多种语言和语音特性，可以应用于不同语言的语音处理任务。
- 6) 性能优势：相比于之前的 Wav2Vec 模型，Wav2Vec2 在语音识别、语音合成等任务上取得了更好的性能，具有更高的准确率和更低的误识率。

### 2. Eating Sound Collection 数据集

数据集中包含 20 种不同食物的咀嚼声音，赛题任务是给这些声音数据建模，准确分类。

- 1) 音频来源：音频素材是从“eating sound”这个词的热门搜索结果中选出的 20 个食物类别中收集的。这 20 种食物中的每一种都以最高质量下载了 12~14 个饮食音频（总共 246 个视频）。所有这些音频都是在房间内录制的，但具有不同的空间特性（例如房间混响）、食物品种（例如有/没有蔬菜的汉堡）、记录质量和饮食行为。
- 2) 剪辑收集数据：使用 Logic Pro X 从收集的视频中剪切出吃东西的声音片段。对于每个视频，所有进食声音以外的声音都被删除，避免包含谈话、餐具和包装声音。重复长时间咀嚼的声音被分成更小的碎片。之后，将目标为 1db 的峰值归一化并应用于所有剪辑区域（其中 0 db 代表失真边缘）。每个食物类别产生 279~873 个片段，总共 11141 个片段。

### 三、实验过程



本实验为了方便观察程序运行时的状态，采用 jupyter notebook 进行实验；

#### 1. 准备数据集

- 1) 从 kaggle 网站上下载 Eating Sound Collection 数据集；
- 2) 删除掉有损坏的文件以及不存在的文件路径；
- 3) 随机打乱数据；
- 4) 构造训练集和测试集，这里设置训练集和测试集的比例为 4:1；
- 5) 最后查看以下数据集中各类别的样本数（如下图）。

```
grapes: 579, chips: 719, candied_fruits: 806, cabbage: 499, gummies: 678,
ice-cream: 727, pizza: 609, fries: 644, jelly: 442, carrots: 660,
aloe: 546, pickles: 872, drinks: 292, chocolate: 290, soup: 278,
burger: 595, wings: 504, ribs: 488, noodles: 411, salmon: 501,
```

#### 2. 数据集预处理

这里处理数据集的主要目的是将数据集处理成模型能够接受的格式。

- 1) 将具体的文件数据导入；
- 2) 对音频数据进行重采样处理，并将 label 进行表示（具体代码如下）；


```
def speech_file_to_array_fn(path):
    speech_array, sampling_rate = torchaudio.load(path)
    resampler = torchaudio.transforms.Resample(sampling_rate, target_sampling_rate)
    speech = resampler(speech_array).squeeze().numpy()
    return speech


def label_to_id(label, label_list):
    if len(label_list) > 0:
        return label_list.index(label) if label in label_list else -1
    return label
```

3) 使用 map 函数分别对训练集和测试集进行处理，处理代码和结果如下：

```
train_dataset = train_dataset.map(preprocess_function, batch_size=10,
                                   batched=True, num_proc=16)
eval_dataset = eval_dataset.map(preprocess_function, batch_size=10,
                                 batched=True, num_proc=16)
```

✓ 1m 55.2s

Map (num\_proc=4): 100%  1000/1000 [01:36<00:00, 25.24 examples/s]

Map (num\_proc=4): 100%  200/200 [00:18<00:00, 15.73 examples/s]

### 3. 模型构建

- 1) 首先确定使用的预训练 wav2vec2 模型，这里使用 huggingface hub 上的 facebook/wav2vec2-base-100k-voxpopuli 模型；
- 2) 为了适配我们这里的分类任务，我们在 Wav2Vec2 模型之后添加一个用于分类任务的头部，具体结构如下所示。这个层由两个线性层构成，目的是将输入从 hidden 维度映射到分类标签的数量上，并且每个层都采用了 dropout 正则化处理；

```
class Wav2Vec2ClassificationHead(nn.Module):
    """Head for wav2vec classification task."""
    def __init__(self, config):
        super().__init__()
        self.dense = nn.Linear(config.hidden_size, config.hidden_size)
        self.dropout = nn.Dropout(config.final_dropout)
        self.out_proj = nn.Linear(config.hidden_size, config.num_labels)

    def forward(self, features, **kwargs):
        x = features
        x = self.dropout(x)
        x = self.dense(x)
        x = torch.tanh(x)
        x = self.dropout(x)
        x = self.out_proj(x)
        return x
```

- 3) 接下来是模型的整体结构，如下所示。模型整体采用了一个 wav2vec2 层

和刚刚构建的分类层；

```
class Wav2Vec2ForSpeechClassification(Wav2Vec2PreTrainedModel):
    def __init__(self, config):
        super().__init__(config)
        self.num_labels = config.num_labels
        self.pooling_mode = config.pooling_mode
        self.config = config

        self.wav2vec2 = Wav2Vec2Model(config)
        self.classifier = Wav2Vec2ClassificationHead(config)
```

4) Merge\_strategy: 这里采用平均策略。

```
def merged_strategy(
    self,
    hidden_states,
    mode="mean"
):
    if mode == "mean":
        outputs = torch.mean(hidden_states, dim=1)
    elif mode == "sum":
        outputs = torch.sum(hidden_states, dim=1)
    elif mode == "max":
        outputs = torch.max(hidden_states, dim=1)[0]
    else:
        raise Exception(
            "The pooling method hasn't been defined! "
            "Your pooling mode must be one of these ['mean', 'sum', 'max']")

    return outputs
```

## 4. 模型训练

数据经过处理后,我们就可以开始设置 training pipeline 了。我们将利用 trainer 进行训练,在训练之前,需要进行以下步骤:

- 1) 定义 data collator。与大多数 NLP 模型相比, XLSR-Wav2Vec2 的输入长度远大于输出长度。例如,输入长度为 50000 的样本的输出长度不超过 100。考虑到输入大小较大,动态填充训练批次会更有效,这意味着所有训练样本只能填充到在此批次中的最长长度,而不是整体最长的样本中。因此,微调 XLSR-Wav2Vec2 需要一个特殊的 padding data collator。Data collator 代码如下所示,具体来说:
  - a) input\_features 和 label\_features 分别提取了特征和标签数据,这些数据将用于填充;
  - b) self.feature\_extractor.pad 方法用于对输入特征进行填充,根据给定的填充策略和最大长度进行填充操作,并返回填充后的张量数据;
  - c) 将填充后的输入特征保存在批次数据中,并根据标签数据的类型创建相



应的张量，然后保存在批次数据中；

```
class DataCollatorCTCWithPadding:
    """...
    feature_extractor: Wav2Vec2FeatureExtractor
    padding: Union[bool, str] = True
    max_length: Optional[int] = None
    max_length_labels: Optional[int] = None
    pad_to_multiple_of: Optional[int] = None
    pad_to_multiple_of_labels: Optional[int] = None

    def __call__(self, features: List[Dict[str, Union[List[int], torch.Tensor]]]) -> Dict[str, torch.Tensor]:
        input_features = [{"input_values": feature["input_values"]} for feature in features]
        label_features = [feature["labels"] for feature in features]
        d_type = torch.long if isinstance(label_features[0], int) else torch.float
        batch = self.feature_extractor.pad(
            input_features,
            padding=self.padding,
            max_length=self.max_length,
            pad_to_multiple_of=self.pad_to_multiple_of,
            return_tensors="pt",
        )
        batch["labels"] = torch.tensor(label_features, dtype=d_type)
        return batch
```

- 2) 评估指标。在训练过程中，应该根据错误率来评估模型。我们应该相应地定义一个“compute\_metrics”函数。这里我们将使用准确率进行分类，使用MSE进行回归；

```
def compute_metrics(p: EvalPrediction):
    preds = p.predictions[0] if isinstance(p.predictions, tuple) else p.predictions
    preds = np.squeeze(preds) if is_regression else np.argmax(preds, axis=1)

    if is_regression:
        return {"mse": ((preds - p.label_ids) ** 2).mean().item()}
    else:
        return {"accuracy": (preds == p.label_ids).astype(np.float32).mean().item()}
```

- 3) 加载预训练模型。我们需要加载预训练的检查点并正确配置它以进行训练；

```
model = Wav2Vec2ForSpeechClassification.from_pretrained(
    model_name_or_path,
    config=config,
)
```

- 4) 定义训练配置。具体配置如下所示；

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./content/wav2vec2-base-100k-eating-sound-collection",
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    gradient_accumulation_steps=2,
    evaluation_strategy="steps",
    num_train_epochs=1.0,
    # fp16=True,
    save_steps=10,
    eval_steps=10,
    logging_steps=10,
    learning_rate=1e-4,
    save_total_limit=2,
)
```



## 5) 开始训练。模型训练结果如下所示。

[62/62 05:43, Epoch 0/1]

Step	Training Loss	Validation Loss	Accuracy
10	2.991500	2.982868	0.050000
20	2.954800	2.982940	0.065000
30	2.999700	2.982800	0.065000
40	3.000500	2.979535	0.085000
50	2.974100	2.978915	0.085000
60	2.941200	2.978330	0.080000

TrainOutput(global\_step=62, training\_loss=2.9760935844913607, metrics={'train\_runtime': 357.8558, 'train\_samples\_per\_second': 2.794, 'train\_steps\_per\_second': 0.173, 'total\_flos': 6.102377894503526e+16, 'train\_loss': 2.9760935844913607, 'epoch': 0.99})

## 5. 模型评价

导入保存的 checkpoint，以备评价步骤：

导入测试数据集并对数据集进行预处理：

进行评价，评价结果如下所示：

	precision	recall	f1-score	support
aloe	0.99	0.87	0.93	109
burger	1.00	0.48	0.65	119
cabbage	0.91	0.95	0.93	100
candied_fruits	0.96	0.99	0.98	161
carrots	0.98	0.98	0.98	132
chips	1.00	0.97	0.98	144
chocolate	0.85	0.98	0.91	58
drinks	1.00	0.98	0.99	58
fries	0.99	0.88	0.93	129
grapes	0.98	0.97	0.98	116
gummies	0.93	0.95	0.94	136
ice-cream	0.97	0.99	0.98	145
jelly	0.91	0.95	0.93	88
noodles	0.88	0.91	0.90	82
pickles	0.98	1.00	0.99	174
pizza	0.75	0.99	0.85	122
ribs	0.89	0.89	0.89	98
salmon	0.74	0.96	0.84	100
soup	0.98	0.93	0.95	56
wings	0.87	0.84	0.85	101
accuracy			0.93	2228
macro avg	0.93	0.92	0.92	2228
weighted avg	0.93	0.93	0.92	2228

## 四、总结

### 1. 请总结本次实验的收获

这次实验我尝试复现了 wav2vec2 模型，完整的进行了一次连续语音识别的实验，从准备数据集到数据集的预处理，从模型的构建到模型的训练，再到模型的评价，对 wav2vec2 模型有了深刻的了解，并对使用大模型来进行语音识别的任务有了深刻的了解，收获颇丰。

### 2. 请给出对本次实验内容的建议

无