Speech Emotion Detection and Analysis

Helen Chan
Travis Ebesu
Caleb Fujimori

COEN296: Natural Language Processing
Prof. Ming-Hwa Wang
School of Engineering
Department of Computer Engineering
Santa Clara University

Table of Contents

Abstract

With the increased popularity of Smart Home technologies, more is expected in Natural Language Processing. We propose a emotion-sensing module with only speech signal as input to detect and analyze the user's emotion based on his or her pitch and volume in the speech sample. This modal uses Convolutional Neural Networks (CNN) to classify emotion from speech.

Introduction

The ultimate goal of our project is to create an emotion-sensing module to be used in Smart Home systems for enriching the user experience and evaluating the user's mental health.This module will detect and analyze emotions in speech samples. Emotion is portrayed in intonation, pitch, volume, and word choice in human. Even with the same sentence, emotions can be ambiguous. Therefore, it is still an area with great potential. In our project, we are focusing on detecting emotions according to pitch and volume. This project is inspired by topics learnt from our class syllabus on text processing, speech recognition, signal processing, and grammar ambiguity.

Many in the industry are detecting emotions from biometrics collected from wearable technologies, which we believe to be not as accessible as microphones. Using pure speech signals, this module can be incorporated into Smart Home or Phone system, which can be shared with a large group of users with just one device around.

In terms of uniqueness of our algorithm and output generation, based on our research, many studies use machine learning methods like Support Vector Machine (SVM) or Recurrent Neural Network (RNN). We decided to use CNN instead because of its improved speed and similar accuracy.

Theoretical Bases and Literature Review

Classically, speech processing has been done by extracting low level descriptors from a voice signal and handcrafting a model. Gu et al. used tuned filters to classify 5 emotions with 94% accuracy[4]. However, these filters were used for a specific dataset of speakers from a specific region. To use this model for another set of speakers, the filters must be re-tuned by experts.

With the rise of machine learning, the features are extracted from a training set, and a model is generated from that data. Wang and Hu extracted 10 features from voice signals and used a SVM to classify voice samples into 7 categories, achieving 85% accuracy [8].

Another common machine learning technique for audio processing is the neural network, which has proven to be more accurate than linear classifiers such as SVM as more data becomes available. Since speech is a time domain signal, it naturally fits the structure of a RNN. Mirsamadi et al showed that an RNN could achieve 64% accuracy in classifying speech into four emotions[6]. Han et al. used an RNN to achieve 66% accuracy in detecting stress from a voice signal[5]. While an RNN intuitively matches speech processing, training is much more costly than training other types of neural networks. In particular, CNN capture local relationships while being faster to train than RNNs.

Abdelwahab and Busso proposed using a Domain Adversarial Neural Network which can more easily be executed on datasets different than the training data[1]. However these adversarial networks are also slow to train.

We cannot directly compare the accuracy between all these results because they all used different datasets, but the higher accuracy of the SVM than the RNNs indicates that explicitly modeling the time relationship of the speech may not be necessary. Recent work has shown that state-of-the-art results on sequential tasks can be obtained with convolutional or non-recurrent architectures (Vaswani et al., 2017) (Gehring, Auli, Grangier, Yarats, & Dauphin, 2017). Furthermore, the use of CNNs will allow for highly parallelizable computation rather than a slow conditional RNN. Therefore, we propose the use of a CNN to classify emotion from speech. Results from others in this area have shown that learning from extracted features can be more accurate, but in other fields, such as Computer Vision, research has shown that training on raw data can be just as, if not more effective as the feature extraction is “learned” by the network. We will train on the raw time series data, frequency domain data, and extracted features.

Using a CNN will greatly reduce training time compared to other RNN approaches, and scale to large datasets better than the SVM or tuned filter based approaches.

Hypothesis

We hypothesize that recognition of emotions in speech are not heavily dependent on a fully conditional probability of the entire speech waveform. Rather a sliding window on the audio will be able to detect strong enough features. Hence, we propose to use a convolutional neural network which will make training much faster, easing development and reducing resources spent on training.

Methodology

Dataset

We will use the IEMOCAP dataset[1] which consists of interactions of male and female actors with improvised and scripted sessions of speech consisting of at least three different annotators labeling one of the following emotions: angry, happy, sad, neutral, frustrated, excited, fearful, surprised, disgusted, other.

Algorithm Design

We will solve the problem of classifying emotions for a given utterance of speech using a Convolutional Neural Network. Specifically, the max-time delay neural network variant (Collobert & Weston, 2008). A given waveform of speech will be encoded with a convolution to capture nonlinear local temporal features.

Language: Python
Tools/Libraries: Keras with TensorFlow backend.
How to generate output:
We will evaluate the performance of our model with respect to baseline models such as logistic regression and support vector machines. If time permits we will also explore the various trade offs between RNNs and CNNs.

How to test against hypotheses: We will use classification accuracy.

[1] "IEMOCAP- Home - USC/Sail - University of Southern California." https://sail.usc.edu/iemocap/. Accessed 4 Nov. 2018.

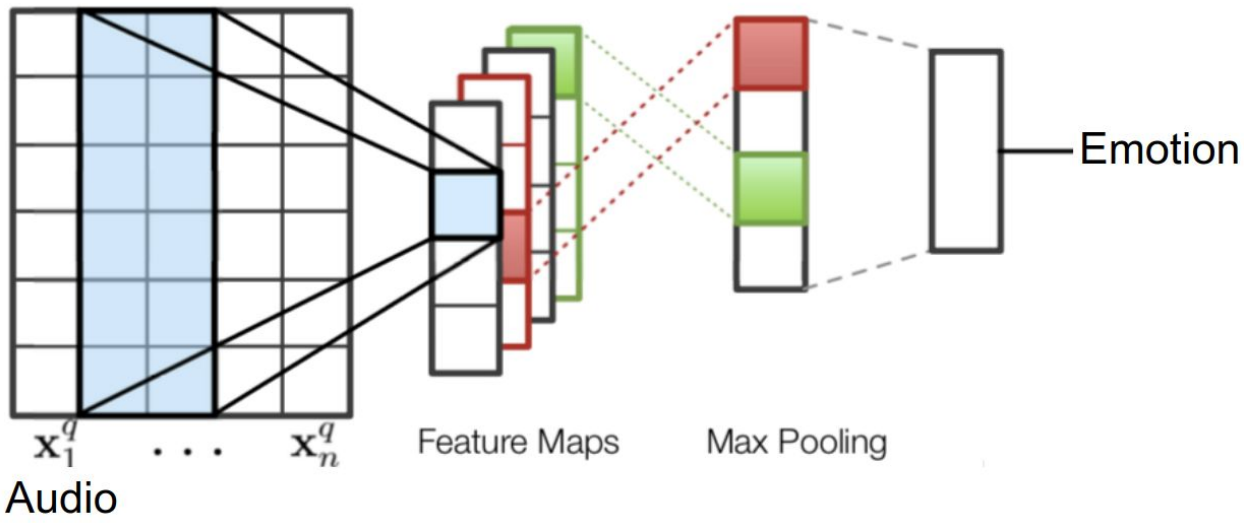
Implementation

Code

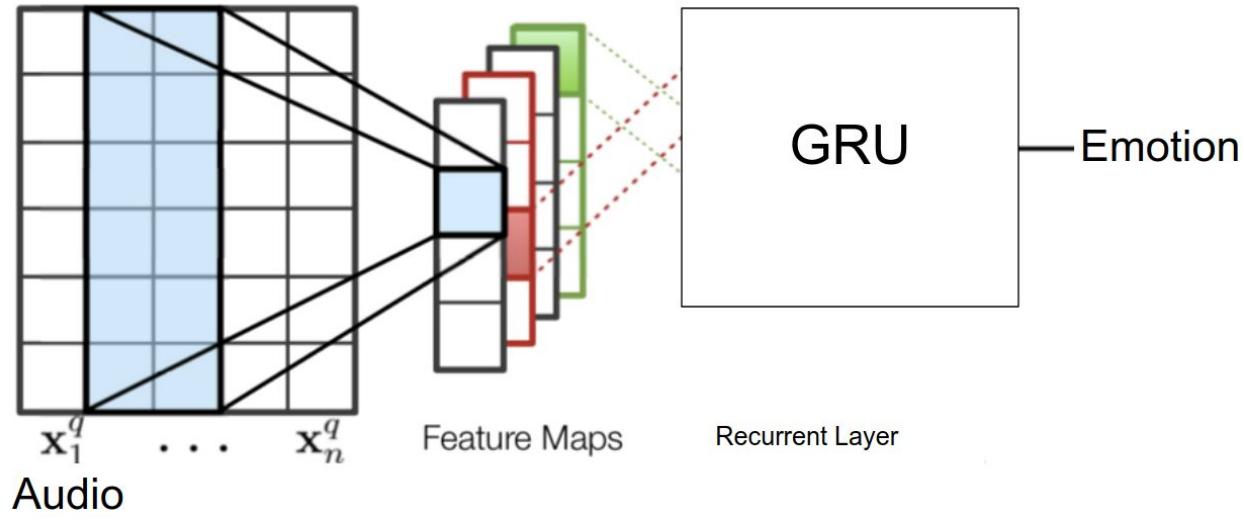
We used the Keras framework with Tensorflow as the backend to code the Neural Nets and scikit-learn for the Logistic Regression and SVM.

Design document and flow chart

CNN Architecture



GRU Architecture



Comparison of methods

- RNN - A recurrent neural network with a fully connected layer applied at each timestep and a GRU
- CNN - Max time delay neural network with 16 filters of kernel size 3. A max pooling window of 9 is applied to feature maps obtained followed by a fully connected layer.
- Logistic Regression - A standard logistic regression model was fit to the data where the temporal speech features were pooled by taking the max over its dimension.
- Support Vector Machine - A standard support vector machine with a linear kernel was applied in a One Vs All fashion to handle the multi-class nature of the problem.

Data Analysis and Discussion

Output Analysis

Model	**Accuracy**
CNN	42.86%
RNN	N/A
Logistic Regression	28.57%
SVM	28.57%

Compare Output against Hypothesis

Our implementation of a GRU network did not work, so the accuracy cannot be compared, but each epoch of the CNN took less than a second to train, while the GRU took about 7 seconds, so our hypothesis that using a CNN would reduce training time was correct.

Discussion

The SVM and Logistic Regression baseline methods perform similarly. This could be due to the fact that the input features are max-pooled in time and hence they obtain the same feature vectors. Furthermore, as both are linear models the ability to correctly classify the speech is limited. The CNN model outperforms SVM and logistic regression particularly due to its ability to capture temporal dependencies over a given timespan.

We are not sure why the GRU failed. We tried many configurations with different activation functions, optimizers, regularizers and surrounding layers, but none of them produced good results. The failure may be due to insufficient support from a small dataset.

Conclusions and Recommendations

Summary and conclusions

In summary, a CNN can perform better than some classifiers such as SVM or logistic regression, and is much faster to train than a GRU.

Recommendations for future studies

In the future, improving the RNN architecture and working with larger datasets would allow for better results. We also used relatively simple architectures for both categories of NN, so further comparison of more advanced architectures is needed to compare both accuracy and training resources.

Bibliography

[1] Mohammed Abdelwahab and Carlos Busso. 2018. Domain Adversarial for Acoustic

Emotion Recognition. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* 26, 12

(December 2018), 2423-2435.

[2] Collobert, R., & Weston, J. (2008). A unified architecture for natural language

processing: Deep neural networks with multitask learning. Paper presented at the *Proceedings of the 25th International Conference on Machine Learning,* 160-167.

[3] Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *arXiv Preprint arXiv:1705.03122,*

[4] Yu Gu, Eric Postma, and Hai-Xiang Lin. 2015. Vocal Emotion Recognition with Log-Gabor Filters. In *Proceedings of the 5th International Workshop on Audio/Visual Emotion Challenge (AVEC '15)*. ACM, New York, NY, USA, 25-31.

[5] Hyewon Han, Kyunggeun Byun, and Hong-Goo Kang. 2018. A Deep Learning-based Stress Detection Algorithm with Speech Signal. In *Proceedings of the 2018 Workshop on Audio-Visual Scene Understanding for Immersive Multimedia (AVSU'18)*. ACM, New York, NY, USA, 11-15.

[6] Mirsamadi, Seyedmahdad & Barsoum, Emad & Zhang, Cha. (2017). Automatic Speech Emotion Recognition Using Recurrent Neural Networks with Local Attention. *10.1109/ICASSP.2017.7952552.*

[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention is all you need. Paper presented at the *Advances in Neural Information Processing Systems,* 5998-6008.

[8] Yan Wang and Weiping Hu. 2018. Speech Emotion Recognition Based on Improved MFCC. In *Proceedings of the 2nd International Conference on Computer Science and Application Engineering (CSAE '18)*. ACM, New York, NY, USA, Article 88, 7 pages

Appendices

Program source code with documentation

#emotionClassification.py

```
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '3'
os.environ['KERAS_BACKEND'] = 'tensorflow'
from sklearn.linear_model import LogisticRegression

import sklearn
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
import numpy as np
import tensorflow as tf
import random
random.seed(0)
tf.set_random_seed(0)
np.random.seed(0)

# set a constant random seed for comparable results
Y_SHAPE = 3
N_LABELS = 6
N_FEATURES = 34
LEN_SENTENCE = 25
LEN_WORD = 60

data = np.load("train.npz")
x = data['x']
y = data['y']
# x.shape
x = x.reshape(x.shape[0], N_FEATURES, -1)
y_hot = tf.keras.utils.to_categorical(y, num_classes=N_LABELS)
val_data = np.load("valid.npz")
val_x = val_data['x']
val_y = val_data['y']
```

```
val_x = val_x.reshape(val_x.shape[0], N_FEATURES, -1)
val_y_hot = tf.keras.utils.to_categorical(val_y, num_classes=N_LABELS)


def cnn():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Conv1D(16, 3, activation='relu',
                                     input_shape=(N_FEATURES, 1500)))
    model.add(tf.keras.layers.MaxPool1D(8))
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(N_LABELS))
    model.add(tf.keras.layers.Activation('softmax'))
    model.compile('adam', 'categorical_crossentropy',
metrics=['accuracy'])

    history = model.fit(x, y_hot, epochs=50, verbose=2,
validation_data=(val_x, val_y_hot))
    return history

def gru():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Permute((2,1), input_shape =
(N_FEATURES, LEN_SENTENCE * LEN_WORD)))
    l2Reg = tf.keras.regularizers.l2(0.01)

model.add(tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(16,
activation='relu', kernel_regularizer=l2Reg, bias_regularizer=l2Reg)))
    model.add(tf.keras.layers.Dropout(0.9))
    model.add(tf.keras.layers.GRU(N_LABELS, activation='relu',
kernel_regularizer=l2Reg, recurrent_regularizer=l2Reg,
bias_regularizer=l2Reg))
    model.add(tf.keras.layers.Activation('softmax'))
    model.compile(optimizer='Adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    history = model.fit(x, y_hot, epochs=50, verbose=2,
validation_data=(val_x, val_y_hot))
    return history
```

```
def lg_svm():
    model = LogisticRegression().fit(x.max(2), y)
    predict = model.predict(val_x.max(2))
    print('Logistic Regression Accuracy')
    print(sklearn.metrics.accuracy_score(predict, val_y))

    model = SVC(kernel='linear')
    model = OneVsRestClassifier(model)
    model.fit(x.max(2), y)
    predict = model.predict(val_x.max(2))
    print('SVM Accuracy')
    print(sklearn.metrics.accuracy_score(predict, val_y))

if __name__=='__main__':
    h = cnn()
    print('CNN accuracy:')
    print(h.history['val_acc'][-1])
    lg_svm()
    h = gru()
    print('GRU accuracy:')
    print(h.history['val_acc'][-1])
```

Readme

Our program requires tensorflow > 1.4 which can be installed by first installing anaconda from https://www.anaconda.com/download/.

Tensorflow can then be installed with:
$conda install tensorflow

The test and train data can be found at:
https://github.com/chf2117/emotionClassification

Download train.npz and valid.npz and put them in the same directory as emotionClassification.py then run the program as:
$python emotionClassification.py

Input/ Output listing

$python emotionClassification.py

```
0.2857142857142857
0.2857142857142857
Train on 62 samples, validate on 7 samples
Epoch 1/50
 - 4s - loss: 4.3592 - acc: 0.1774 - val_loss: 3.8241 - val_acc: 0.4286
Epoch 2/50
 - 0s - loss: 6.1037 - acc: 0.1129 - val_loss: 3.8984 - val_acc: 0.4286
Epoch 3/50
 - 0s - loss: 5.8958 - acc: 0.1613 - val_loss: 3.9012 - val_acc: 0.2857
Epoch 4/50
 - 0s - loss: 4.1182 - acc: 0.2097 - val_loss: 4.1826 - val_acc: 0.2857
Epoch 5/50
 - 0s - loss: 3.8878 - acc: 0.1613 - val_loss: 4.4766 - val_acc: 0.2857
Epoch 6/50
 - 0s - loss: 4.8856 - acc: 0.1774 - val_loss: 4.7254 - val_acc: 0.2857
Epoch 7/50
 - 0s - loss: 4.2569 - acc: 0.1935 - val_loss: 4.9760 - val_acc: 0.2857
Epoch 8/50
 - 0s - loss: 3.7973 - acc: 0.1290 - val_loss: 5.1718 - val_acc: 0.2857
Epoch 9/50
 - 0s - loss: 4.9107 - acc: 0.1613 - val_loss: 5.3347 - val_acc: 0.2857
Epoch 10/50
 - 0s - loss: 4.2882 - acc: 0.1613 - val_loss: 5.4458 - val_acc: 0.2857
Epoch 11/50
 - 0s - loss: 4.6679 - acc: 0.1774 - val_loss: 5.5110 - val_acc: 0.2857
Epoch 12/50
 - 0s - loss: 5.2415 - acc: 0.1935 - val_loss: 5.5953 - val_acc: 0.2857
Epoch 13/50
 - 0s - loss: 4.7104 - acc: 0.1129 - val_loss: 5.7097 - val_acc: 0.2857
Epoch 14/50
 - 0s - loss: 3.7289 - acc: 0.1613 - val_loss: 5.8048 - val_acc: 0.2857
Epoch 15/50
 - 0s - loss: 3.7667 - acc: 0.1613 - val_loss: 5.7606 - val_acc: 0.2857
Epoch 16/50
 - 0s - loss: 4.7166 - acc: 0.1774 - val_loss: 5.8457 - val_acc: 0.2857
Epoch 17/50
 - 0s - loss: 4.6852 - acc: 0.1452 - val_loss: 6.0550 - val_acc: 0.2857
Epoch 18/50
 - 0s - loss: 2.8564 - acc: 0.1452 - val_loss: 6.1820 - val_acc: 0.4286
```

Epoch 19/50
 - 0s - loss: 3.1221 - acc: 0.1613 - val_loss: 6.1366 - val_acc: 0.4286
Epoch 20/50
 - 0s - loss: 4.6116 - acc: 0.2097 - val_loss: 6.1982 - val_acc: 0.4286
Epoch 21/50
 - 0s - loss: 3.6334 - acc: 0.1935 - val_loss: 6.1344 - val_acc: 0.4286
Epoch 22/50
 - 0s - loss: 3.1255 - acc: 0.1613 - val_loss: 6.1293 - val_acc: 0.4286
Epoch 23/50
 - 0s - loss: 3.7325 - acc: 0.1774 - val_loss: 6.1612 - val_acc: 0.4286
Epoch 24/50
 - 0s - loss: 4.4974 - acc: 0.1129 - val_loss: 6.1834 - val_acc: 0.4286
Epoch 25/50
 - 0s - loss: 5.0983 - acc: 0.1613 - val_loss: 6.1849 - val_acc: 0.4286
Epoch 26/50
 - 0s - loss: 3.7952 - acc: 0.1452 - val_loss: 6.1768 - val_acc: 0.4286
Epoch 27/50
 - 0s - loss: 3.7837 - acc: 0.1935 - val_loss: 6.1596 - val_acc: 0.4286
Epoch 28/50
 - 0s - loss: 3.7889 - acc: 0.1290 - val_loss: 6.1461 - val_acc: 0.4286
Epoch 29/50
 - 0s - loss: 3.6361 - acc: 0.1290 - val_loss: 6.2393 - val_acc: 0.4286
Epoch 30/50
 - 0s - loss: 3.6965 - acc: 0.1774 - val_loss: 6.2722 - val_acc: 0.4286
Epoch 31/50
 - 0s - loss: 4.6424 - acc: 0.1452 - val_loss: 6.2575 - val_acc: 0.4286
Epoch 32/50
 - 0s - loss: 4.7173 - acc: 0.1290 - val_loss: 6.2461 - val_acc: 0.4286
Epoch 33/50
 - 0s - loss: 4.4262 - acc: 0.1452 - val_loss: 6.2349 - val_acc: 0.4286
Epoch 34/50
 - 0s - loss: 3.1356 - acc: 0.1613 - val_loss: 6.2662 - val_acc: 0.4286
Epoch 35/50
 - 0s - loss: 3.9161 - acc: 0.1452 - val_loss: 6.2959 - val_acc: 0.4286
Epoch 36/50
 - 0s - loss: 4.2455 - acc: 0.1290 - val_loss: 6.3220 - val_acc: 0.4286
Epoch 37/50
 - 0s - loss: 4.7273 - acc: 0.1290 - val_loss: 6.3437 - val_acc: 0.4286
Epoch 38/50
 - 0s - loss: 4.4894 - acc: 0.0968 - val_loss: 6.3476 - val_acc: 0.4286
Epoch 39/50
 - 0s - loss: 3.7860 - acc: 0.1774 - val_loss: 6.3510 - val_acc: 0.4286
Epoch 40/50

- 0s - loss: 3.3067 - acc: 0.1613 - val_loss: 6.3546 - val_acc: 0.4286
Epoch 41/50
- 0s - loss: 4.1955 - acc: 0.2097 - val_loss: 6.2628 - val_acc: 0.4286
Epoch 42/50
- 0s - loss: 4.0090 - acc: 0.0968 - val_loss: 6.1994 - val_acc: 0.4286
Epoch 43/50
- 0s - loss: 4.4169 - acc: 0.1613 - val_loss: 6.1602 - val_acc: 0.4286
Epoch 44/50
- 0s - loss: 5.7774 - acc: 0.1290 - val_loss: 6.1195 - val_acc: 0.4286
Epoch 45/50
- 0s - loss: 3.3043 - acc: 0.1935 - val_loss: 6.0806 - val_acc: 0.4286
Epoch 46/50
- 0s - loss: 4.0351 - acc: 0.1613 - val_loss: 6.0491 - val_acc: 0.4286
Epoch 47/50
- 0s - loss: 3.7153 - acc: 0.2097 - val_loss: 5.9780 - val_acc: 0.4286
Epoch 48/50
- 0s - loss: 3.0653 - acc: 0.1774 - val_loss: 5.8493 - val_acc: 0.4286
Epoch 49/50
- 0s - loss: 3.6765 - acc: 0.1613 - val_loss: 5.6734 - val_acc: 0.4286
Epoch 50/50
- 0s - loss: 4.7502 - acc: 0.1290 - val_loss: 5.5554 - val_acc: 0.4286

GRU training
Train on 62 samples, validate on 7 samples
Epoch 1/50
2018-12-04 05:44:49.366761: I tensorflow/core/common_runtime/process_util.cc:69] Creating new thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best performance.
- 8s - loss: 2.0880 - acc: 0.2419 - val_loss: 2.0848 - val_acc: 0.2857
Epoch 2/50
- 7s - loss: 2.0839 - acc: 0.1613 - val_loss: 2.0801 - val_acc: 0.2857
Epoch 3/50
- 7s - loss: 2.0793 - acc: 0.1613 - val_loss: 2.0754 - val_acc: 0.2857
Epoch 4/50
- 7s - loss: 2.0746 - acc: 0.1613 - val_loss: 2.0706 - val_acc: 0.2857
Epoch 5/50
- 7s - loss: 2.0699 - acc: 0.1613 - val_loss: 2.0659 - val_acc: 0.2857
Epoch 6/50
- 7s - loss: 2.0652 - acc: 0.1613 - val_loss: 2.0612 - val_acc: 0.2857
Epoch 7/50
- 7s - loss: 2.0605 - acc: 0.1613 - val_loss: 2.0566 - val_acc: 0.2857

Epoch 8/50
 - 7s - loss: 2.0559 - acc: 0.1613 - val_loss: 2.0521 - val_acc: 0.2857
Epoch 9/50
 - 7s - loss: 2.0514 - acc: 0.1613 - val_loss: 2.0476 - val_acc: 0.2857
Epoch 10/50
 - 7s - loss: 2.0469 - acc: 0.1613 - val_loss: 2.0432 - val_acc: 0.2857
Epoch 11/50
 - 7s - loss: 2.0424 - acc: 0.1613 - val_loss: 2.0388 - val_acc: 0.2857
Epoch 12/50
 - 7s - loss: 2.0381 - acc: 0.1613 - val_loss: 2.0345 - val_acc: 0.2857
Epoch 13/50
 - 7s - loss: 2.0338 - acc: 0.1613 - val_loss: 2.0303 - val_acc: 0.2857
Epoch 14/50
 - 7s - loss: 2.0295 - acc: 0.1613 - val_loss: 2.0262 - val_acc: 0.2857
Epoch 15/50
 - 7s - loss: 2.0253 - acc: 0.1613 - val_loss: 2.0221 - val_acc: 0.2857
Epoch 16/50
 - 7s - loss: 2.0212 - acc: 0.1613 - val_loss: 2.0181 - val_acc: 0.2857
Epoch 17/50
 - 7s - loss: 2.0172 - acc: 0.1613 - val_loss: 2.0141 - val_acc: 0.2857
Epoch 18/50
 - 7s - loss: 2.0132 - acc: 0.1613 - val_loss: 2.0102 - val_acc: 0.2857
Epoch 19/50
 - 7s - loss: 2.0093 - acc: 0.1613 - val_loss: 2.0064 - val_acc: 0.2857
Epoch 20/50
 - 7s - loss: 2.0055 - acc: 0.1613 - val_loss: 2.0026 - val_acc: 0.0000e+00
Epoch 21/50
 - 7s - loss: 2.0017 - acc: 0.1613 - val_loss: 1.9989 - val_acc: 0.0000e+00
Epoch 22/50
 - 7s - loss: 1.9980 - acc: 0.1613 - val_loss: 1.9952 - val_acc: 0.0000e+00
Epoch 23/50
 - 7s - loss: 1.9944 - acc: 0.1613 - val_loss: 1.9917 - val_acc: 0.0000e+00
Epoch 24/50
 - 7s - loss: 1.9908 - acc: 0.1613 - val_loss: 1.9881 - val_acc: 0.0000e+00
Epoch 25/50
 - 7s - loss: 1.9873 - acc: 0.1613 - val_loss: 1.9849 - val_acc: 0.0000e+00
Epoch 26/50
 - 7s - loss: 1.9838 - acc: 0.2097 - val_loss: 1.9817 - val_acc: 0.0000e+00
Epoch 27/50
 - 7s - loss: 1.9803 - acc: 0.2097 - val_loss: 1.9785 - val_acc: 0.0000e+00
Epoch 28/50
 - 7s - loss: 1.9769 - acc: 0.2097 - val_loss: 1.9755 - val_acc: 0.0000e+00
Epoch 29/50

 - 7s - loss: 1.9735 - acc: 0.2097 - val_loss: 1.9727 - val_acc: 0.0000e+00
Epoch 30/50
 - 7s - loss: 1.9703 - acc: 0.2097 - val_loss: 1.9699 - val_acc: 0.0000e+00
Epoch 31/50
 - 7s - loss: 1.9670 - acc: 0.2097 - val_loss: 1.9671 - val_acc: 0.0000e+00
Epoch 32/50
 - 7s - loss: 1.9638 - acc: 0.2097 - val_loss: 1.9644 - val_acc: 0.0000e+00
Epoch 33/50
 - 7s - loss: 1.9606 - acc: 0.2097 - val_loss: 1.9617 - val_acc: 0.0000e+00
Epoch 34/50
 - 8s - loss: 1.9575 - acc: 0.2097 - val_loss: 1.9591 - val_acc: 0.0000e+00
Epoch 35/50
 - 9s - loss: 1.9545 - acc: 0.2097 - val_loss: 1.9566 - val_acc: 0.0000e+00
Epoch 36/50
 - 9s - loss: 1.9514 - acc: 0.2097 - val_loss: 1.9540 - val_acc: 0.0000e+00
Epoch 37/50
 - 9s - loss: 1.9486 - acc: 0.2097 - val_loss: 1.9516 - val_acc: 0.0000e+00
Epoch 38/50
 - 13s - loss: 1.9456 - acc: 0.2097 - val_loss: 1.9492 - val_acc: 0.0000e+00
Epoch 39/50
 - 18s - loss: 1.9428 - acc: 0.2097 - val_loss: 1.9468 - val_acc: 0.0000e+00
Epoch 40/50
 - 18s - loss: 1.9400 - acc: 0.2097 - val_loss: 1.9445 - val_acc: 0.0000e+00
Epoch 41/50
 - 18s - loss: 1.9373 - acc: 0.2097 - val_loss: 1.9422 - val_acc: 0.0000e+00
Epoch 42/50
 - 18s - loss: 1.9346 - acc: 0.2097 - val_loss: 1.9399 - val_acc: 0.0000e+00
Epoch 43/50
 - 18s - loss: 1.9319 - acc: 0.2097 - val_loss: 1.9377 - val_acc: 0.0000e+00
Epoch 44/50
 - 18s - loss: 1.9293 - acc: 0.2097 - val_loss: 1.9356 - val_acc: 0.0000e+00
Epoch 45/50
 - 18s - loss: 1.9268 - acc: 0.2097 - val_loss: 1.9335 - val_acc: 0.0000e+00
Epoch 46/50
 - 17s - loss: 1.9242 - acc: 0.2097 - val_loss: 1.9314 - val_acc: 0.0000e+00
Epoch 47/50
 - 18s - loss: 1.9218 - acc: 0.2097 - val_loss: 1.9294 - val_acc: 0.0000e+00
Epoch 48/50
 - 18s - loss: 1.9194 - acc: 0.2097 - val_loss: 1.9274 - val_acc: 0.0000e+00
Epoch 49/50
 - 18s - loss: 1.9169 - acc: 0.2097 - val_loss: 1.9254 - val_acc: 0.0000e+00
Epoch 50/50
 - 18s - loss: 1.9146 - acc: 0.2097 - val_loss: 1.9235 - val_acc: 0.0000e+00