


```
In [1]: 1 # Keras
2 import keras
3 from keras import regularizers
4 from keras.preprocessing import sequence
5 from keras.preprocessing.text import Tokenizer
6 # from keras.preprocessing.sequence import pad_sequences
7 from keras.models import Sequential, Model, model_from_json
8 from keras.layers import Dense, Embedding, LSTM
9 from keras.layers import Input, Flatten, Dropout, Activation, BatchNormalization
10 from keras.layers import Conv1D, MaxPooling1D, AveragePooling1D
11 from keras.utils import np_utils, to_categorical
12 from keras.callbacks import (EarlyStopping, LearningRateScheduler,
13                               ModelCheckpoint, TensorBoard, ReduceLROnPlateau)
14 from keras import losses, models, optimizers
15 from keras.activations import relu, softmax
16 from keras.layers import (Convolution2D, GlobalAveragePooling2D, BatchNormalization, Flatten, Dropout,
17                             GlobalMaxPool2D, MaxPool2D, concatenate, Activation, Input, Dense)
18
19 # sklearn
20 from sklearn.metrics import confusion_matrix, accuracy_score
21 from sklearn.model_selection import train_test_split
22 from sklearn.preprocessing import LabelEncoder
23
24 # Other
25 from tqdm import tqdm, tqdm_pandas
26 import scipy
27 from scipy.stats import skew
28 import librosa
29 import librosa.display
30 import json
31 import numpy as np
32 import matplotlib.pyplot as plt
33 import tensorflow as tf
34 from matplotlib.pyplot import specgram
35 import pandas as pd
36 import seaborn as sns
37 import glob
38 import os
39 import sys
40 import IPython.display as ipd # To play sound in the notebook
41 import warnings
```

```
42 # ignore warnings
43 if not sys.warnoptions:
44     warnings.simplefilter("ignore")
```

```
In [2]: 1 ref = pd.read_csv("Data_path.csv")
        2 ref.head()
```

Out[2]:

	labels	source	path
0	male_surprise	RAVDESS	data/RAVDESS/Actor_01/03-01-08-02-02-01-01.wav
1	male_surprise	RAVDESS	data/RAVDESS/Actor_01/03-01-08-01-01-01-01.wav
2	male_angry	RAVDESS	data/RAVDESS/Actor_01/03-01-05-01-02-01-01.wav
3	male_fear	RAVDESS	data/RAVDESS/Actor_01/03-01-06-01-02-02-01.wav
4	male_fear	RAVDESS	data/RAVDESS/Actor_01/03-01-06-02-01-02-01.wav

In [3]:

```

1  '''
2  1. Data Augmentation method
3  '''
4  def speedNpitch(data):
5      """
6      Speed and Pitch Tuning.
7      """
8      # you can change low and high here
9      length_change = np.random.uniform(low=0.8, high = 1)
10     speed_fac = 1.2 / length_change # try changing 1.0 to 2.0 ... =D
11     tmp = np.interp(np.arange(0,len(data),speed_fac),np.arange(0,len(data)),data)
12     minlen = min(data.shape[0], tmp.shape[0])
13     data *= 0
14     data[0:minlen] = tmp[0:minlen]
15     return data
16
17 '''
18 2. Extracting the MFCC feature as an image (Matrix format).
19 '''
20 def prepare_data(df, n, aug, mfcc):
21     X = np.empty(shape=(df.shape[0], n, 216, 1))
22     input_length = sampling_rate * audio_duration
23
24     cnt = 0
25     for fname in tqdm(df.path):
26         file_path = fname
27         data, _ = librosa.load(file_path, sr=sampling_rate
28                                ,res_type="kaiser_fast"
29                                ,duration=2.5
30                                ,offset=0.5
31                                )
32
33         # Random offset / Padding
34         if len(data) > input_length:
35             max_offset = len(data) - input_length
36             offset = np.random.randint(max_offset)
37             data = data[offset:(input_length+offset)]
38         else:
39             if input_length > len(data):
40                 max_offset = input_length - len(data)
41                 offset = np.random.randint(max_offset)

```

```

42         else:
43             offset = 0
44             data = np.pad(data, (offset, int(input_length) - len(data) - offset), "constant")
45
46         # Augmentation?
47         if aug == 1:
48             data = speedNpitch(data)
49
50         # which feature?
51         if mfcc == 1:
52             # MFCC extraction
53             MFCC = librosa.feature.mfcc(data, sr=sampling_rate, n_mfcc=n_mfcc)
54             MFCC = np.expand_dims(MFCC, axis=-1)
55             X[cnt,] = MFCC
56
57         else:
58             # Log-melspectrogram
59             melspec = librosa.feature.melspectrogram(data, n_mels = n_melspec)
60             logspec = librosa.amplitude_to_db(melspec)
61             logspec = np.expand_dims(logspec, axis=-1)
62             X[cnt,] = logspec
63
64         cnt += 1
65
66     return X
67
68 '''
69 3. Confusion matrix plot
70 '''
71
72 def print_confusion_matrix(confusion_matrix, class_names, figsize = (10,7), fontsize=14):
73     '''Prints a confusion matrix, as returned by sklearn.metrics.confusion_matrix, as a heatmap.
74
75     Arguments
76     -----
77     confusion_matrix: numpy.ndarray
78         The numpy.ndarray object returned from a call to sklearn.metrics.confusion_matrix.
79         Similarly constructed ndarrays can also be used.
80     class_names: list
81         An ordered list of class names, in the order they index the given confusion matrix.
82     figsize: tuple
83         A 2-long tuple, the first value determining the horizontal size of the ouputed figure,

```

```

84         the second determining the vertical size. Defaults to (10,7).
85     fontsize: int
86         Font size for axes labels. Defaults to 14.
87
88     Returns
89     -----
90     matplotlib.figure.Figure
91         The resulting confusion matrix figure
92     '''
93     df_cm = pd.DataFrame(
94         confusion_matrix, index=class_names, columns=class_names,
95     )
96     fig = plt.figure(figsize=figsize)
97     try:
98         heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
99     except ValueError:
100         raise ValueError("Confusion matrix values must be integers.")
101
102     heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=fontsize)
103     heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', fontsize=fontsize)
104     plt.ylabel('True label')
105     plt.xlabel('Predicted label')
106
107
108
109     '''
110 # 4. Create the 2D CNN model
111 '''
112 def get_2d_conv_model(n):
113     ''' Create a standard deep 2D convolutional neural network'''
114     nclass = 14
115     inp = Input(shape=(n,216,1)) #2D matrix of 30 MFCC bands by 216 audio length.
116     x = Convolution2D(32, (4,10), padding="same")(inp)
117     x = BatchNormalization()(x)
118     x = Activation("relu")(x)
119     x = MaxPool2D()(x)
120     x = Dropout(rate=0.2)(x)
121
122     x = Convolution2D(32, (4,10), padding="same")(x)
123     x = BatchNormalization()(x)
124     x = Activation("relu")(x)
125     x = MaxPool2D()(x)

```

```

126     x = Dropout(rate=0.2)(x)
127
128     x = Convolution2D(32, (4,10), padding="same")(x)
129     x = BatchNormalization()(x)
130     x = Activation("relu")(x)
131     x = MaxPool2D()(x)
132     x = Dropout(rate=0.2)(x)
133
134     x = Convolution2D(32, (4,10), padding="same")(x)
135     x = BatchNormalization()(x)
136     x = Activation("relu")(x)
137     x = MaxPool2D()(x)
138     x = Dropout(rate=0.2)(x)
139
140     x = Flatten()(x)
141     x = Dense(64)(x)
142     x = Dropout(rate=0.2)(x)
143     x = BatchNormalization()(x)
144     x = Activation("relu")(x)
145     x = Dropout(rate=0.2)(x)
146
147     out = Dense(nclass, activation=softmax)(x)
148     model = models.Model(inputs=inp, outputs=out)
149
150     opt = optimizers.Adam(0.00001)
151     # opt = keras.optimizers.RMSprop(lr=0.00001, decay=1e-6)
152     model.compile(optimizer=opt, loss=losses.categorical_crossentropy, metrics=['acc'])
153     model.summary()
154     return model
155
156     '''
157     # 5. Other functions
158     '''
159     class get_results:
160         '''
161         We're going to create a class (blueprint template) for generating the results based on the various mode
162         So instead of repeating the functions each time, we assign the results into on object with its associat
163         depending on each combination:
164             1) MFCC with no augmentation
165             2) MFCC with augmentation
166             3) Logmelspec with no augmentation
167             4) Logmelspec with augmentation

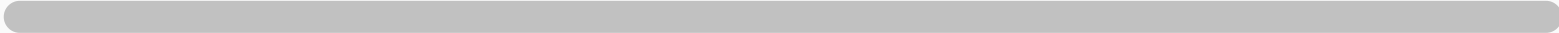
```

```
168     '''
169
170     def __init__(self, model_history, model ,X_test, y_test, labels):
171         self.model_history = model_history
172         self.model = model
173         self.X_test = X_test
174         self.y_test = y_test
175         self.labels = labels
176
177     def create_plot(self, model_history):
178         '''Check the logloss of both train and validation, make sure they are close and have plateau'''
179         plt.plot(model_history.history['loss'])
180         plt.plot(model_history.history['val_loss'])
181         plt.title('model loss')
182         plt.ylabel('loss')
183         plt.xlabel('epoch')
184         plt.legend(['train', 'test'], loc='upper left')
185         plt.show()
186
187     def create_results(self, model):
188         '''predict on test set and get accuracy results'''
189         opt = optimizers.Adam(0.00001)
190         model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
191         score = model.evaluate(X_test, y_test, verbose=0)
192         print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))
193
194     def confusion_results(self, X_test, y_test, labels, model):
195         '''plot confusion matrix results'''
196         preds = model.predict(X_test,
197                               batch_size=16,
198                               verbose=2)
199         preds=preds.argmax(axis=1)
200         preds = preds.astype(int).flatten()
201         preds = (lb.inverse_transform((preds)))
202
203         actual = y_test.argmax(axis=1)
204         actual = actual.astype(int).flatten()
205         actual = (lb.inverse_transform((actual)))
206
207         classes = labels
208         classes.sort()
209
```



```
210     c = confusion_matrix(actual, preds)
211     print_confusion_matrix(c, class_names = classes)
212
213     def accuracy_results_gender(self, X_test, y_test, labels, model):
214         '''Print out the accuracy score and confusion matrix heat map of the Gender classification results'''
215
216         preds = model.predict(X_test,
217                               batch_size=16,
218                               verbose=2)
219         preds=preds.argmax(axis=1)
220         preds = preds.astype(int).flatten()
221         preds = (lb.inverse_transform((preds)))
222
223         actual = y_test.argmax(axis=1)
224         actual = actual.astype(int).flatten()
225         actual = (lb.inverse_transform((actual)))
226
227         # print(accuracy_score(actual, preds))
228
229         actual = pd.DataFrame(actual).replace({'female_angry': 'female'
230         , 'female_disgust': 'female'
231         , 'female_fear': 'female'
232         , 'female_happy': 'female'
233         , 'female_sad': 'female'
234         , 'female_surprise': 'female'
235         , 'female_neutral': 'female'
236         , 'male_angry': 'male'
237         , 'male_fear': 'male'
238         , 'male_happy': 'male'
239         , 'male_sad': 'male'
240         , 'male_surprise': 'male'
241         , 'male_neutral': 'male'
242         , 'male_disgust': 'male'
243         })
244         preds = pd.DataFrame(preds).replace({'female_angry': 'female'
245         , 'female_disgust': 'female'
246         , 'female_fear': 'female'
247         , 'female_happy': 'female'
248         , 'female_sad': 'female'
249         , 'female_surprise': 'female'
250         , 'female_neutral': 'female'
251         , 'male_angry': 'male'
```

```
252         , 'male_fear':'male'
253         , 'male_happy':'male'
254         , 'male_sad':'male'
255         , 'male_surprise':'male'
256         , 'male_neutral':'male'
257         , 'male_disgust':'male'
258     })
259
260     classes = actual.loc[:,0].unique()
261     classes.sort()
262
263     c = confusion_matrix(actual, preds)
264     print(accuracy_score(actual, preds))
265     print_confusion_matrix(c, class_names = classes)
```



```

In [4]: 1 sampling_rate=44100
        2 audio_duration=2.5
        3 n_mfcc = 30
        4 mfcc = prepare_data(ref, n = n_mfcc, aug = 0, mfcc = 1)
        5
        6 # Split between train and test
        7 X_train, X_test, y_train, y_test = train_test_split(mfcc
        8                                                         , ref.labels
        9                                                         , test_size=0.25
       10                                                         , shuffle=True
       11                                                         , random_state=42
       12                                                         )
       13
       14
       15 # one hot encode the target
       16 lb = LabelEncoder()
       17 y_train = np_utils.to_categorical(lb.fit_transform(y_train))
       18 y_test = np_utils.to_categorical(lb.fit_transform(y_test))
       19
       20 # Normalization as per the standard NN process
       21 mean = np.mean(X_train, axis=0)
       22 std = np.std(X_train, axis=0)
       23
       24 X_train = (X_train - mean)/std
       25 X_test = (X_test - mean)/std
       26
       27 # Build CNN model
       28 model = get_2d_conv_model(n=n_mfcc)
       29 model_history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
       30                          batch_size=16, verbose = 2, epochs=200)

```

100%|██| 1440/1440 [01:19<00:00, 18.17it/s]

2022-10-19 16:23:40.363802: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:306] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA support.

2022-10-19 16:23:40.364006: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:272] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)

Metal device set to: Apple M2

Model: "model"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 30, 216, 1)]	0
conv2d (Conv2D)	(None, 30, 216, 32)	1312
batch_normalization (Batch Normalization)	(None, 30, 216, 32)	128
activation (Activation)	(None, 30, 216, 32)	0
max_pooling2d (MaxPooling2D)	(None, 15, 108, 32)	0
dropout (Dropout)	(None, 15, 108, 32)	0
conv2d_1 (Conv2D)	(None, 15, 108, 32)	40992
batch_normalization_1 (Batch Normalization)	(None, 15, 108, 32)	128
activation_1 (Activation)	(None, 15, 108, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 7, 54, 32)	0
dropout_1 (Dropout)	(None, 7, 54, 32)	0
conv2d_2 (Conv2D)	(None, 7, 54, 32)	40992
batch_normalization_2 (Batch Normalization)	(None, 7, 54, 32)	128
activation_2 (Activation)	(None, 7, 54, 32)	0
max_pooling2d_2 (MaxPooling2D)	(None, 3, 27, 32)	0
dropout_2 (Dropout)	(None, 3, 27, 32)	0

conv2d_3 (Conv2D)	(None, 3, 27, 32)	40992
batch_normalization_3 (Batch Normalization)	(None, 3, 27, 32)	128
activation_3 (Activation)	(None, 3, 27, 32)	0
max_pooling2d_3 (MaxPooling2D)	(None, 1, 13, 32)	0
dropout_3 (Dropout)	(None, 1, 13, 32)	0
flatten (Flatten)	(None, 416)	0
dense (Dense)	(None, 64)	26688
dropout_4 (Dropout)	(None, 64)	0
batch_normalization_4 (Batch Normalization)	(None, 64)	256
activation_4 (Activation)	(None, 64)	0
dropout_5 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 14)	910

=====

Total params: 152,654

Trainable params: 152,270

Non-trainable params: 384

Epoch 1/200

2022-10-19 16:23:40.664888: W tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU frequency: 0 Hz

2022-10-19 16:23:41.111004: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_type GPU is enabled.

2022-10-19 16:23:43.695678: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_type GPU is enabled.

68/68 - 3s - loss: 3.1903 - acc: 0.0713 - val_loss: 2.6797 - val_acc: 0.0639 - 3s/epoch - 49ms/step

```
Epoch 2/200
68/68 - 3s - loss: 3.1312 - acc: 0.0694 - val_loss: 2.7173 - val_acc: 0.0611 - 3s/epoch - 38ms/step
Epoch 3/200
68/68 - 3s - loss: 3.0762 - acc: 0.0787 - val_loss: 2.7263 - val_acc: 0.0583 - 3s/epoch - 38ms/step
Epoch 4/200
68/68 - 3s - loss: 3.0099 - acc: 0.0833 - val_loss: 2.7247 - val_acc: 0.0611 - 3s/epoch - 41ms/step
Epoch 5/200
68/68 - 4s - loss: 2.9177 - acc: 0.0806 - val_loss: 2.7155 - val_acc: 0.0611 - 4s/epoch - 55ms/step
Epoch 6/200
68/68 - 3s - loss: 2.9106 - acc: 0.0889 - val_loss: 2.7076 - val_acc: 0.0639 - 3s/epoch - 45ms/step
Epoch 7/200
68/68 - 3s - loss: 2.9116 - acc: 0.0889 - val_loss: 2.6966 - val_acc: 0.0639 - 3s/epoch - 45ms/step
Epoch 8/200
68/68 - 3s - loss: 2.8416 - acc: 0.1009 - val_loss: 2.6825 - val_acc: 0.0639 - 3s/epoch - 41ms/step
Epoch 9/200
68/68 - 3s - loss: 2.8112 - acc: 0.1000 - val_loss: 2.6722 - val_acc: 0.0667 - 3s/epoch - 39ms/step
Epoch 10/200
68/68 - 3s - loss: 2.7714 - acc: 0.1046 - val_loss: 2.6630 - val_acc: 0.0667 - 3s/epoch - 39ms/step
Epoch 11/200
68/68 - 3s - loss: 2.7093 - acc: 0.1111 - val_loss: 2.6601 - val_acc: 0.0667 - 3s/epoch - 41ms/step
Epoch 12/200
68/68 - 3s - loss: 2.6695 - acc: 0.1287 - val_loss: 2.6561 - val_acc: 0.0694 - 3s/epoch - 38ms/step
Epoch 13/200
68/68 - 3s - loss: 2.6505 - acc: 0.1370 - val_loss: 2.6530 - val_acc: 0.0722 - 3s/epoch - 37ms/step
Epoch 14/200
68/68 - 3s - loss: 2.6485 - acc: 0.1213 - val_loss: 2.6456 - val_acc: 0.0694 - 3s/epoch - 38ms/step
Epoch 15/200
68/68 - 3s - loss: 2.6095 - acc: 0.1398 - val_loss: 2.6414 - val_acc: 0.0778 - 3s/epoch - 38ms/step
Epoch 16/200
68/68 - 3s - loss: 2.6344 - acc: 0.1528 - val_loss: 2.6322 - val_acc: 0.0833 - 3s/epoch - 38ms/step
Epoch 17/200
68/68 - 3s - loss: 2.5611 - acc: 0.1574 - val_loss: 2.6239 - val_acc: 0.0861 - 3s/epoch - 38ms/step
Epoch 18/200
68/68 - 3s - loss: 2.5765 - acc: 0.1463 - val_loss: 2.6200 - val_acc: 0.0917 - 3s/epoch - 38ms/step
Epoch 19/200
68/68 - 3s - loss: 2.5524 - acc: 0.1722 - val_loss: 2.6143 - val_acc: 0.0917 - 3s/epoch - 38ms/step
Epoch 20/200
68/68 - 3s - loss: 2.5482 - acc: 0.1611 - val_loss: 2.6052 - val_acc: 0.0944 - 3s/epoch - 44ms/step
Epoch 21/200
68/68 - 3s - loss: 2.5077 - acc: 0.1769 - val_loss: 2.5986 - val_acc: 0.0972 - 3s/epoch - 42ms/step
Epoch 22/200
68/68 - 3s - loss: 2.5353 - acc: 0.1833 - val_loss: 2.5877 - val_acc: 0.1056 - 3s/epoch - 38ms/step
```

```
Epoch 23/200
68/68 - 3s - loss: 2.4982 - acc: 0.1954 - val_loss: 2.5833 - val_acc: 0.1139 - 3s/epoch - 38ms/step
Epoch 24/200
68/68 - 3s - loss: 2.4399 - acc: 0.2130 - val_loss: 2.5750 - val_acc: 0.1167 - 3s/epoch - 40ms/step
Epoch 25/200
68/68 - 3s - loss: 2.4727 - acc: 0.1861 - val_loss: 2.5686 - val_acc: 0.1167 - 3s/epoch - 39ms/step
Epoch 26/200
68/68 - 3s - loss: 2.4296 - acc: 0.1981 - val_loss: 2.5605 - val_acc: 0.1194 - 3s/epoch - 39ms/step
Epoch 27/200
68/68 - 3s - loss: 2.4380 - acc: 0.2019 - val_loss: 2.5516 - val_acc: 0.1361 - 3s/epoch - 38ms/step
Epoch 28/200
68/68 - 3s - loss: 2.4142 - acc: 0.2185 - val_loss: 2.5491 - val_acc: 0.1361 - 3s/epoch - 41ms/step
Epoch 29/200
68/68 - 3s - loss: 2.4140 - acc: 0.2083 - val_loss: 2.5392 - val_acc: 0.1472 - 3s/epoch - 40ms/step
Epoch 30/200
68/68 - 3s - loss: 2.3898 - acc: 0.2296 - val_loss: 2.5298 - val_acc: 0.1500 - 3s/epoch - 38ms/step
Epoch 31/200
68/68 - 3s - loss: 2.3860 - acc: 0.2389 - val_loss: 2.5236 - val_acc: 0.1667 - 3s/epoch - 38ms/step
Epoch 32/200
68/68 - 3s - loss: 2.3695 - acc: 0.2481 - val_loss: 2.5207 - val_acc: 0.1667 - 3s/epoch - 39ms/step
Epoch 33/200
68/68 - 3s - loss: 2.3349 - acc: 0.2435 - val_loss: 2.5139 - val_acc: 0.1639 - 3s/epoch - 38ms/step
Epoch 34/200
68/68 - 3s - loss: 2.3515 - acc: 0.2509 - val_loss: 2.5087 - val_acc: 0.1722 - 3s/epoch - 38ms/step
Epoch 35/200
68/68 - 3s - loss: 2.3308 - acc: 0.2491 - val_loss: 2.4996 - val_acc: 0.1833 - 3s/epoch - 43ms/step
Epoch 36/200
68/68 - 3s - loss: 2.3485 - acc: 0.2370 - val_loss: 2.4887 - val_acc: 0.1972 - 3s/epoch - 45ms/step
Epoch 37/200
68/68 - 3s - loss: 2.3173 - acc: 0.2509 - val_loss: 2.4885 - val_acc: 0.1806 - 3s/epoch - 46ms/step
Epoch 38/200
68/68 - 3s - loss: 2.2952 - acc: 0.2685 - val_loss: 2.4855 - val_acc: 0.1750 - 3s/epoch - 48ms/step
Epoch 39/200
68/68 - 3s - loss: 2.2663 - acc: 0.2593 - val_loss: 2.4782 - val_acc: 0.1861 - 3s/epoch - 41ms/step
Epoch 40/200
68/68 - 3s - loss: 2.2516 - acc: 0.2593 - val_loss: 2.4739 - val_acc: 0.1972 - 3s/epoch - 40ms/step
Epoch 41/200
68/68 - 3s - loss: 2.2878 - acc: 0.2556 - val_loss: 2.4654 - val_acc: 0.2056 - 3s/epoch - 40ms/step
Epoch 42/200
68/68 - 3s - loss: 2.2704 - acc: 0.2454 - val_loss: 2.4649 - val_acc: 0.1972 - 3s/epoch - 39ms/step
Epoch 43/200
68/68 - 3s - loss: 2.2796 - acc: 0.2481 - val_loss: 2.4567 - val_acc: 0.2083 - 3s/epoch - 38ms/step
```

```
Epoch 44/200
68/68 - 3s - loss: 2.2453 - acc: 0.2759 - val_loss: 2.4517 - val_acc: 0.2000 - 3s/epoch - 40ms/step
Epoch 45/200
68/68 - 3s - loss: 2.2491 - acc: 0.2648 - val_loss: 2.4483 - val_acc: 0.2139 - 3s/epoch - 38ms/step
Epoch 46/200
68/68 - 3s - loss: 2.1880 - acc: 0.2889 - val_loss: 2.4358 - val_acc: 0.2278 - 3s/epoch - 38ms/step
Epoch 47/200
68/68 - 3s - loss: 2.2131 - acc: 0.2685 - val_loss: 2.4305 - val_acc: 0.2361 - 3s/epoch - 38ms/step
Epoch 48/200
68/68 - 3s - loss: 2.1914 - acc: 0.3046 - val_loss: 2.4243 - val_acc: 0.2444 - 3s/epoch - 38ms/step
Epoch 49/200
68/68 - 3s - loss: 2.1587 - acc: 0.2824 - val_loss: 2.4182 - val_acc: 0.2500 - 3s/epoch - 38ms/step
Epoch 50/200
68/68 - 3s - loss: 2.1850 - acc: 0.2907 - val_loss: 2.4161 - val_acc: 0.2472 - 3s/epoch - 38ms/step
Epoch 51/200
68/68 - 3s - loss: 2.2075 - acc: 0.2759 - val_loss: 2.4106 - val_acc: 0.2472 - 3s/epoch - 38ms/step
Epoch 52/200
68/68 - 3s - loss: 2.1738 - acc: 0.2861 - val_loss: 2.4046 - val_acc: 0.2472 - 3s/epoch - 38ms/step
Epoch 53/200
68/68 - 3s - loss: 2.1670 - acc: 0.2889 - val_loss: 2.4005 - val_acc: 0.2472 - 3s/epoch - 38ms/step
Epoch 54/200
68/68 - 3s - loss: 2.1500 - acc: 0.2880 - val_loss: 2.4009 - val_acc: 0.2444 - 3s/epoch - 38ms/step
Epoch 55/200
68/68 - 3s - loss: 2.1503 - acc: 0.3000 - val_loss: 2.3995 - val_acc: 0.2444 - 3s/epoch - 38ms/step
Epoch 56/200
68/68 - 3s - loss: 2.1585 - acc: 0.2981 - val_loss: 2.3906 - val_acc: 0.2556 - 3s/epoch - 38ms/step
Epoch 57/200
68/68 - 3s - loss: 2.1572 - acc: 0.3065 - val_loss: 2.3885 - val_acc: 0.2472 - 3s/epoch - 39ms/step
Epoch 58/200
68/68 - 3s - loss: 2.1524 - acc: 0.3139 - val_loss: 2.3913 - val_acc: 0.2472 - 3s/epoch - 38ms/step
Epoch 59/200
68/68 - 3s - loss: 2.1391 - acc: 0.3009 - val_loss: 2.3833 - val_acc: 0.2528 - 3s/epoch - 37ms/step
Epoch 60/200
68/68 - 3s - loss: 2.0982 - acc: 0.3176 - val_loss: 2.3782 - val_acc: 0.2556 - 3s/epoch - 38ms/step
Epoch 61/200
68/68 - 3s - loss: 2.1108 - acc: 0.3231 - val_loss: 2.3734 - val_acc: 0.2500 - 3s/epoch - 38ms/step
Epoch 62/200
68/68 - 3s - loss: 2.1357 - acc: 0.2981 - val_loss: 2.3627 - val_acc: 0.2500 - 3s/epoch - 38ms/step
Epoch 63/200
68/68 - 3s - loss: 2.1004 - acc: 0.3148 - val_loss: 2.3579 - val_acc: 0.2528 - 3s/epoch - 39ms/step
Epoch 64/200
68/68 - 3s - loss: 2.0625 - acc: 0.3157 - val_loss: 2.3608 - val_acc: 0.2528 - 3s/epoch - 45ms/step
```



```
Epoch 65/200
68/68 - 3s - loss: 2.0845 - acc: 0.3287 - val_loss: 2.3599 - val_acc: 0.2472 - 3s/epoch - 45ms/step
Epoch 66/200
68/68 - 3s - loss: 2.1048 - acc: 0.3130 - val_loss: 2.3485 - val_acc: 0.2444 - 3s/epoch - 41ms/step
Epoch 67/200
68/68 - 3s - loss: 2.0884 - acc: 0.3204 - val_loss: 2.3459 - val_acc: 0.2472 - 3s/epoch - 41ms/step
Epoch 68/200
68/68 - 3s - loss: 2.0520 - acc: 0.3574 - val_loss: 2.3435 - val_acc: 0.2444 - 3s/epoch - 45ms/step
Epoch 69/200
68/68 - 3s - loss: 2.0852 - acc: 0.3139 - val_loss: 2.3414 - val_acc: 0.2444 - 3s/epoch - 49ms/step
Epoch 70/200
68/68 - 3s - loss: 2.0541 - acc: 0.3296 - val_loss: 2.3360 - val_acc: 0.2472 - 3s/epoch - 43ms/step
Epoch 71/200
68/68 - 3s - loss: 2.0661 - acc: 0.3343 - val_loss: 2.3322 - val_acc: 0.2444 - 3s/epoch - 46ms/step
Epoch 72/200
68/68 - 3s - loss: 2.0530 - acc: 0.3389 - val_loss: 2.3300 - val_acc: 0.2500 - 3s/epoch - 48ms/step
Epoch 73/200
68/68 - 3s - loss: 2.0507 - acc: 0.3407 - val_loss: 2.3276 - val_acc: 0.2472 - 3s/epoch - 48ms/step
Epoch 74/200
68/68 - 3s - loss: 2.0482 - acc: 0.3444 - val_loss: 2.3232 - val_acc: 0.2444 - 3s/epoch - 48ms/step
Epoch 75/200
68/68 - 3s - loss: 2.0110 - acc: 0.3472 - val_loss: 2.3248 - val_acc: 0.2444 - 3s/epoch - 48ms/step
Epoch 76/200
68/68 - 3s - loss: 2.0012 - acc: 0.3509 - val_loss: 2.3199 - val_acc: 0.2417 - 3s/epoch - 44ms/step
Epoch 77/200
68/68 - 3s - loss: 1.9883 - acc: 0.3556 - val_loss: 2.3119 - val_acc: 0.2472 - 3s/epoch - 39ms/step
Epoch 78/200
68/68 - 3s - loss: 2.0171 - acc: 0.3435 - val_loss: 2.3113 - val_acc: 0.2444 - 3s/epoch - 39ms/step
Epoch 79/200
68/68 - 3s - loss: 1.9976 - acc: 0.3620 - val_loss: 2.3088 - val_acc: 0.2444 - 3s/epoch - 44ms/step
Epoch 80/200
68/68 - 3s - loss: 1.9998 - acc: 0.3574 - val_loss: 2.3072 - val_acc: 0.2417 - 3s/epoch - 40ms/step
Epoch 81/200
68/68 - 3s - loss: 1.9748 - acc: 0.3806 - val_loss: 2.2977 - val_acc: 0.2472 - 3s/epoch - 39ms/step
Epoch 82/200
68/68 - 3s - loss: 2.0120 - acc: 0.3519 - val_loss: 2.2967 - val_acc: 0.2472 - 3s/epoch - 39ms/step
Epoch 83/200
68/68 - 3s - loss: 1.9830 - acc: 0.3611 - val_loss: 2.2965 - val_acc: 0.2472 - 3s/epoch - 38ms/step
Epoch 84/200
68/68 - 3s - loss: 1.9677 - acc: 0.3667 - val_loss: 2.2879 - val_acc: 0.2556 - 3s/epoch - 39ms/step
Epoch 85/200
68/68 - 3s - loss: 1.9642 - acc: 0.3537 - val_loss: 2.2867 - val_acc: 0.2500 - 3s/epoch - 38ms/step
```

```
Epoch 86/200
68/68 - 3s - loss: 1.9758 - acc: 0.3463 - val_loss: 2.2838 - val_acc: 0.2528 - 3s/epoch - 38ms/step
Epoch 87/200
68/68 - 3s - loss: 1.9420 - acc: 0.3898 - val_loss: 2.2786 - val_acc: 0.2583 - 3s/epoch - 39ms/step
Epoch 88/200
68/68 - 3s - loss: 1.9594 - acc: 0.3583 - val_loss: 2.2839 - val_acc: 0.2472 - 3s/epoch - 38ms/step
Epoch 89/200
68/68 - 3s - loss: 1.9637 - acc: 0.3750 - val_loss: 2.2836 - val_acc: 0.2472 - 3s/epoch - 40ms/step
Epoch 90/200
68/68 - 3s - loss: 1.9184 - acc: 0.3935 - val_loss: 2.2834 - val_acc: 0.2444 - 3s/epoch - 43ms/step
Epoch 91/200
68/68 - 3s - loss: 1.9454 - acc: 0.3824 - val_loss: 2.2806 - val_acc: 0.2417 - 3s/epoch - 50ms/step
Epoch 92/200
68/68 - 3s - loss: 1.9133 - acc: 0.3741 - val_loss: 2.2700 - val_acc: 0.2528 - 3s/epoch - 48ms/step
Epoch 93/200
68/68 - 3s - loss: 1.8767 - acc: 0.4009 - val_loss: 2.2633 - val_acc: 0.2500 - 3s/epoch - 38ms/step
Epoch 94/200
68/68 - 3s - loss: 1.9278 - acc: 0.3750 - val_loss: 2.2599 - val_acc: 0.2500 - 3s/epoch - 40ms/step
Epoch 95/200
68/68 - 3s - loss: 1.9075 - acc: 0.3954 - val_loss: 2.2580 - val_acc: 0.2500 - 3s/epoch - 42ms/step
Epoch 96/200
68/68 - 3s - loss: 1.9156 - acc: 0.3620 - val_loss: 2.2596 - val_acc: 0.2500 - 3s/epoch - 46ms/step
Epoch 97/200
68/68 - 3s - loss: 1.8862 - acc: 0.3880 - val_loss: 2.2530 - val_acc: 0.2556 - 3s/epoch - 42ms/step
Epoch 98/200
68/68 - 3s - loss: 1.8789 - acc: 0.3954 - val_loss: 2.2509 - val_acc: 0.2472 - 3s/epoch - 45ms/step
Epoch 99/200
68/68 - 3s - loss: 1.8816 - acc: 0.4065 - val_loss: 2.2499 - val_acc: 0.2472 - 3s/epoch - 41ms/step
Epoch 100/200
68/68 - 3s - loss: 1.8932 - acc: 0.3778 - val_loss: 2.2482 - val_acc: 0.2472 - 3s/epoch - 40ms/step
Epoch 101/200
68/68 - 3s - loss: 1.8908 - acc: 0.3815 - val_loss: 2.2461 - val_acc: 0.2444 - 3s/epoch - 41ms/step
Epoch 102/200
68/68 - 3s - loss: 1.8736 - acc: 0.3944 - val_loss: 2.2423 - val_acc: 0.2528 - 3s/epoch - 38ms/step
Epoch 103/200
68/68 - 3s - loss: 1.8642 - acc: 0.4046 - val_loss: 2.2349 - val_acc: 0.2583 - 3s/epoch - 41ms/step
Epoch 104/200
68/68 - 3s - loss: 1.8448 - acc: 0.4130 - val_loss: 2.2325 - val_acc: 0.2556 - 3s/epoch - 45ms/step
Epoch 105/200
68/68 - 3s - loss: 1.8397 - acc: 0.3935 - val_loss: 2.2372 - val_acc: 0.2500 - 3s/epoch - 39ms/step
Epoch 106/200
68/68 - 3s - loss: 1.8491 - acc: 0.4019 - val_loss: 2.2305 - val_acc: 0.2472 - 3s/epoch - 46ms/step
```

```
Epoch 107/200
68/68 - 3s - loss: 1.8639 - acc: 0.4139 - val_loss: 2.2182 - val_acc: 0.2556 - 3s/epoch - 46ms/step
Epoch 108/200
68/68 - 4s - loss: 1.8444 - acc: 0.4083 - val_loss: 2.2192 - val_acc: 0.2583 - 4s/epoch - 57ms/step
Epoch 109/200
68/68 - 3s - loss: 1.8184 - acc: 0.4148 - val_loss: 2.2218 - val_acc: 0.2583 - 3s/epoch - 39ms/step
Epoch 110/200
68/68 - 3s - loss: 1.8336 - acc: 0.4278 - val_loss: 2.2213 - val_acc: 0.2528 - 3s/epoch - 39ms/step
Epoch 111/200
68/68 - 3s - loss: 1.8374 - acc: 0.4315 - val_loss: 2.2266 - val_acc: 0.2444 - 3s/epoch - 39ms/step
Epoch 112/200
68/68 - 3s - loss: 1.8385 - acc: 0.3926 - val_loss: 2.2216 - val_acc: 0.2500 - 3s/epoch - 40ms/step
Epoch 113/200
68/68 - 3s - loss: 1.7892 - acc: 0.4352 - val_loss: 2.2147 - val_acc: 0.2528 - 3s/epoch - 43ms/step
Epoch 114/200
68/68 - 3s - loss: 1.7990 - acc: 0.4398 - val_loss: 2.2122 - val_acc: 0.2556 - 3s/epoch - 46ms/step
Epoch 115/200
68/68 - 3s - loss: 1.8164 - acc: 0.4046 - val_loss: 2.2058 - val_acc: 0.2556 - 3s/epoch - 45ms/step
Epoch 116/200
68/68 - 3s - loss: 1.8384 - acc: 0.4130 - val_loss: 2.2097 - val_acc: 0.2472 - 3s/epoch - 45ms/step
Epoch 117/200
68/68 - 3s - loss: 1.7976 - acc: 0.4204 - val_loss: 2.2065 - val_acc: 0.2500 - 3s/epoch - 40ms/step
Epoch 118/200
68/68 - 3s - loss: 1.7722 - acc: 0.4380 - val_loss: 2.2075 - val_acc: 0.2500 - 3s/epoch - 43ms/step
Epoch 119/200
68/68 - 3s - loss: 1.7660 - acc: 0.4343 - val_loss: 2.2015 - val_acc: 0.2583 - 3s/epoch - 43ms/step
Epoch 120/200
68/68 - 3s - loss: 1.7829 - acc: 0.4296 - val_loss: 2.2048 - val_acc: 0.2500 - 3s/epoch - 43ms/step
Epoch 121/200
68/68 - 3s - loss: 1.7897 - acc: 0.4130 - val_loss: 2.1931 - val_acc: 0.2500 - 3s/epoch - 40ms/step
Epoch 122/200
68/68 - 3s - loss: 1.7546 - acc: 0.4491 - val_loss: 2.1902 - val_acc: 0.2528 - 3s/epoch - 41ms/step
Epoch 123/200
68/68 - 3s - loss: 1.7748 - acc: 0.4194 - val_loss: 2.1947 - val_acc: 0.2472 - 3s/epoch - 39ms/step
Epoch 124/200
68/68 - 3s - loss: 1.7546 - acc: 0.4519 - val_loss: 2.1975 - val_acc: 0.2472 - 3s/epoch - 40ms/step
Epoch 125/200
68/68 - 3s - loss: 1.7360 - acc: 0.4731 - val_loss: 2.1932 - val_acc: 0.2500 - 3s/epoch - 38ms/step
Epoch 126/200
68/68 - 3s - loss: 1.7701 - acc: 0.4380 - val_loss: 2.1895 - val_acc: 0.2500 - 3s/epoch - 38ms/step
Epoch 127/200
68/68 - 3s - loss: 1.7402 - acc: 0.4500 - val_loss: 2.1875 - val_acc: 0.2528 - 3s/epoch - 40ms/step
```

```
Epoch 128/200
68/68 - 3s - loss: 1.7426 - acc: 0.4454 - val_loss: 2.1869 - val_acc: 0.2528 - 3s/epoch - 39ms/step
Epoch 129/200
68/68 - 3s - loss: 1.7479 - acc: 0.4574 - val_loss: 2.1819 - val_acc: 0.2500 - 3s/epoch - 41ms/step
Epoch 130/200
68/68 - 3s - loss: 1.7528 - acc: 0.4491 - val_loss: 2.1876 - val_acc: 0.2500 - 3s/epoch - 39ms/step
Epoch 131/200
68/68 - 3s - loss: 1.7466 - acc: 0.4648 - val_loss: 2.1815 - val_acc: 0.2500 - 3s/epoch - 41ms/step
Epoch 132/200
68/68 - 3s - loss: 1.7087 - acc: 0.4694 - val_loss: 2.1696 - val_acc: 0.2583 - 3s/epoch - 41ms/step
Epoch 133/200
68/68 - 3s - loss: 1.7328 - acc: 0.4481 - val_loss: 2.1705 - val_acc: 0.2556 - 3s/epoch - 42ms/step
Epoch 134/200
68/68 - 3s - loss: 1.6907 - acc: 0.4880 - val_loss: 2.1716 - val_acc: 0.2528 - 3s/epoch - 41ms/step
Epoch 135/200
68/68 - 3s - loss: 1.7077 - acc: 0.4722 - val_loss: 2.1675 - val_acc: 0.2583 - 3s/epoch - 42ms/step
Epoch 136/200
68/68 - 3s - loss: 1.7236 - acc: 0.4657 - val_loss: 2.1634 - val_acc: 0.2611 - 3s/epoch - 41ms/step
Epoch 137/200
68/68 - 3s - loss: 1.6909 - acc: 0.4880 - val_loss: 2.1595 - val_acc: 0.2639 - 3s/epoch - 42ms/step
Epoch 138/200
68/68 - 3s - loss: 1.6976 - acc: 0.4833 - val_loss: 2.1564 - val_acc: 0.2639 - 3s/epoch - 40ms/step
Epoch 139/200
68/68 - 3s - loss: 1.6762 - acc: 0.4917 - val_loss: 2.1618 - val_acc: 0.2611 - 3s/epoch - 40ms/step
Epoch 140/200
68/68 - 3s - loss: 1.6954 - acc: 0.4639 - val_loss: 2.1579 - val_acc: 0.2667 - 3s/epoch - 41ms/step
Epoch 141/200
68/68 - 3s - loss: 1.6684 - acc: 0.4787 - val_loss: 2.1494 - val_acc: 0.2694 - 3s/epoch - 39ms/step
Epoch 142/200
68/68 - 3s - loss: 1.6559 - acc: 0.4731 - val_loss: 2.1463 - val_acc: 0.2722 - 3s/epoch - 40ms/step
Epoch 143/200
68/68 - 3s - loss: 1.6420 - acc: 0.4907 - val_loss: 2.1445 - val_acc: 0.2667 - 3s/epoch - 38ms/step
Epoch 144/200
68/68 - 3s - loss: 1.6403 - acc: 0.4824 - val_loss: 2.1397 - val_acc: 0.2694 - 3s/epoch - 39ms/step
Epoch 145/200
68/68 - 3s - loss: 1.6209 - acc: 0.5083 - val_loss: 2.1417 - val_acc: 0.2667 - 3s/epoch - 42ms/step
Epoch 146/200

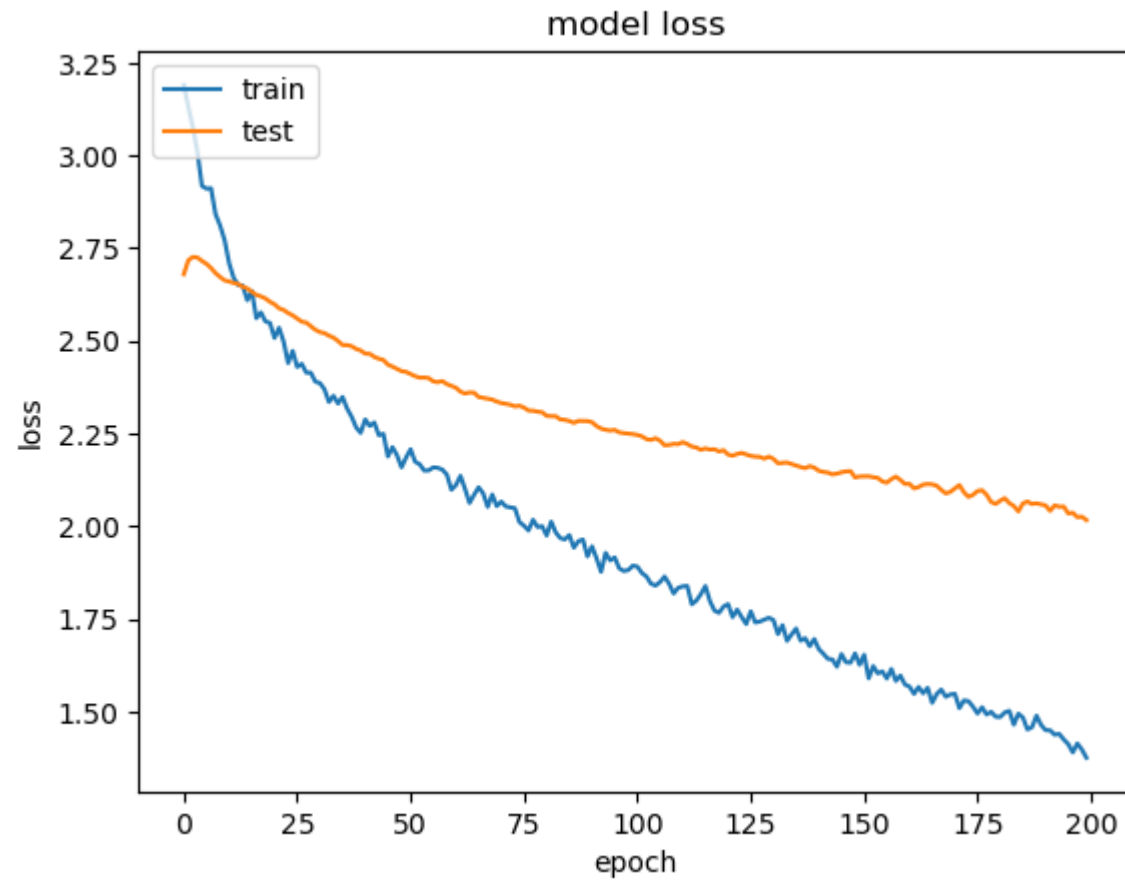
68/68 - 3s - loss: 1.6547 - acc: 0.4815 - val_loss: 2.1452 - val_acc: 0.2694 - 3s/epoch - 43ms/step
Epoch 147/200
68/68 - 3s - loss: 1.6327 - acc: 0.4981 - val_loss: 2.1477 - val_acc: 0.2667 - 3s/epoch - 40ms/step
```

```
Epoch 148/200
68/68 - 3s - loss: 1.6317 - acc: 0.4991 - val_loss: 2.1483 - val_acc: 0.2639 - 3s/epoch - 41ms/step
Epoch 149/200
68/68 - 3s - loss: 1.6565 - acc: 0.4750 - val_loss: 2.1313 - val_acc: 0.2667 - 3s/epoch - 44ms/step
Epoch 150/200
68/68 - 3s - loss: 1.6260 - acc: 0.5102 - val_loss: 2.1341 - val_acc: 0.2639 - 3s/epoch - 41ms/step
Epoch 151/200
68/68 - 3s - loss: 1.6530 - acc: 0.4944 - val_loss: 2.1345 - val_acc: 0.2667 - 3s/epoch - 39ms/step
Epoch 152/200
68/68 - 3s - loss: 1.5894 - acc: 0.5111 - val_loss: 2.1346 - val_acc: 0.2667 - 3s/epoch - 41ms/step
Epoch 153/200
68/68 - 3s - loss: 1.6223 - acc: 0.4852 - val_loss: 2.1320 - val_acc: 0.2611 - 3s/epoch - 39ms/step
Epoch 154/200
68/68 - 3s - loss: 1.6027 - acc: 0.5009 - val_loss: 2.1302 - val_acc: 0.2611 - 3s/epoch - 41ms/step
Epoch 155/200
68/68 - 3s - loss: 1.6081 - acc: 0.4880 - val_loss: 2.1207 - val_acc: 0.2639 - 3s/epoch - 40ms/step
Epoch 156/200
68/68 - 3s - loss: 1.5889 - acc: 0.5139 - val_loss: 2.1172 - val_acc: 0.2667 - 3s/epoch - 41ms/step
Epoch 157/200
68/68 - 3s - loss: 1.6146 - acc: 0.4972 - val_loss: 2.1269 - val_acc: 0.2611 - 3s/epoch - 42ms/step
Epoch 158/200
68/68 - 3s - loss: 1.5824 - acc: 0.5148 - val_loss: 2.1335 - val_acc: 0.2556 - 3s/epoch - 40ms/step
Epoch 159/200
68/68 - 3s - loss: 1.5967 - acc: 0.4944 - val_loss: 2.1245 - val_acc: 0.2694 - 3s/epoch - 40ms/step
Epoch 160/200
68/68 - 3s - loss: 1.5726 - acc: 0.5222 - val_loss: 2.1136 - val_acc: 0.2694 - 3s/epoch - 41ms/step
Epoch 161/200
68/68 - 4s - loss: 1.5670 - acc: 0.5093 - val_loss: 2.1142 - val_acc: 0.2639 - 4s/epoch - 52ms/step
Epoch 162/200
68/68 - 5s - loss: 1.5476 - acc: 0.5444 - val_loss: 2.1019 - val_acc: 0.2778 - 5s/epoch - 70ms/step
Epoch 163/200
68/68 - 4s - loss: 1.5661 - acc: 0.5204 - val_loss: 2.1054 - val_acc: 0.2722 - 4s/epoch - 56ms/step
Epoch 164/200
68/68 - 3s - loss: 1.5489 - acc: 0.4991 - val_loss: 2.1121 - val_acc: 0.2667 - 3s/epoch - 50ms/step
Epoch 165/200
68/68 - 3s - loss: 1.5638 - acc: 0.5269 - val_loss: 2.1137 - val_acc: 0.2722 - 3s/epoch - 42ms/step
Epoch 166/200
68/68 - 3s - loss: 1.5234 - acc: 0.5352 - val_loss: 2.1128 - val_acc: 0.2722 - 3s/epoch - 39ms/step
Epoch 167/200
68/68 - 3s - loss: 1.5469 - acc: 0.5352 - val_loss: 2.1090 - val_acc: 0.2694 - 3s/epoch - 40ms/step
Epoch 168/200
68/68 - 3s - loss: 1.5591 - acc: 0.5157 - val_loss: 2.0978 - val_acc: 0.2750 - 3s/epoch - 46ms/step
```

```
Epoch 169/200
68/68 - 4s - loss: 1.5397 - acc: 0.5417 - val_loss: 2.0883 - val_acc: 0.2806 - 4s/epoch - 54ms/step
Epoch 170/200
68/68 - 3s - loss: 1.5458 - acc: 0.5250 - val_loss: 2.0924 - val_acc: 0.2750 - 3s/epoch - 49ms/step
Epoch 171/200
68/68 - 3s - loss: 1.5476 - acc: 0.5231 - val_loss: 2.1014 - val_acc: 0.2694 - 3s/epoch - 47ms/step
Epoch 172/200
68/68 - 3s - loss: 1.5090 - acc: 0.5417 - val_loss: 2.1109 - val_acc: 0.2611 - 3s/epoch - 51ms/step
Epoch 173/200
68/68 - 3s - loss: 1.5296 - acc: 0.5361 - val_loss: 2.0914 - val_acc: 0.2722 - 3s/epoch - 50ms/step
Epoch 174/200
68/68 - 4s - loss: 1.5276 - acc: 0.5333 - val_loss: 2.0789 - val_acc: 0.2806 - 4s/epoch - 62ms/step
Epoch 175/200
68/68 - 3s - loss: 1.5128 - acc: 0.5481 - val_loss: 2.0844 - val_acc: 0.2750 - 3s/epoch - 42ms/step
Epoch 176/200
68/68 - 4s - loss: 1.4945 - acc: 0.5389 - val_loss: 2.0947 - val_acc: 0.2694 - 4s/epoch - 54ms/step
Epoch 177/200
68/68 - 3s - loss: 1.5113 - acc: 0.5463 - val_loss: 2.0969 - val_acc: 0.2722 - 3s/epoch - 48ms/step
Epoch 178/200
68/68 - 3s - loss: 1.4917 - acc: 0.5426 - val_loss: 2.0847 - val_acc: 0.2694 - 3s/epoch - 45ms/step
Epoch 179/200
68/68 - 3s - loss: 1.4990 - acc: 0.5426 - val_loss: 2.0650 - val_acc: 0.2806 - 3s/epoch - 45ms/step
Epoch 180/200
68/68 - 3s - loss: 1.4858 - acc: 0.5574 - val_loss: 2.0601 - val_acc: 0.2861 - 3s/epoch - 49ms/step
Epoch 181/200
68/68 - 3s - loss: 1.4850 - acc: 0.5472 - val_loss: 2.0685 - val_acc: 0.2778 - 3s/epoch - 43ms/step
Epoch 182/200
68/68 - 3s - loss: 1.4972 - acc: 0.5528 - val_loss: 2.0754 - val_acc: 0.2861 - 3s/epoch - 45ms/step
Epoch 183/200
68/68 - 3s - loss: 1.5015 - acc: 0.5565 - val_loss: 2.0622 - val_acc: 0.2889 - 3s/epoch - 40ms/step
Epoch 184/200
68/68 - 3s - loss: 1.4644 - acc: 0.5565 - val_loss: 2.0543 - val_acc: 0.3000 - 3s/epoch - 50ms/step
Epoch 185/200
68/68 - 3s - loss: 1.4939 - acc: 0.5435 - val_loss: 2.0396 - val_acc: 0.3028 - 3s/epoch - 42ms/step
Epoch 186/200
68/68 - 3s - loss: 1.4836 - acc: 0.5324 - val_loss: 2.0612 - val_acc: 0.2861 - 3s/epoch - 39ms/step
Epoch 187/200
68/68 - 3s - loss: 1.4515 - acc: 0.5759 - val_loss: 2.0664 - val_acc: 0.2833 - 3s/epoch - 50ms/step
Epoch 188/200
68/68 - 3s - loss: 1.4573 - acc: 0.5602 - val_loss: 2.0598 - val_acc: 0.2917 - 3s/epoch - 45ms/step
Epoch 189/200
68/68 - 3s - loss: 1.4885 - acc: 0.5500 - val_loss: 2.0615 - val_acc: 0.2917 - 3s/epoch - 44ms/step
```

```
Epoch 190/200
68/68 - 3s - loss: 1.4638 - acc: 0.5648 - val_loss: 2.0588 - val_acc: 0.2889 - 3s/epoch - 47ms/step
Epoch 191/200
68/68 - 3s - loss: 1.4496 - acc: 0.5657 - val_loss: 2.0559 - val_acc: 0.2861 - 3s/epoch - 44ms/step
Epoch 192/200
68/68 - 3s - loss: 1.4492 - acc: 0.5722 - val_loss: 2.0410 - val_acc: 0.3000 - 3s/epoch - 46ms/step
Epoch 193/200
68/68 - 3s - loss: 1.4373 - acc: 0.5546 - val_loss: 2.0559 - val_acc: 0.2889 - 3s/epoch - 45ms/step
Epoch 194/200
68/68 - 3s - loss: 1.4395 - acc: 0.5667 - val_loss: 2.0523 - val_acc: 0.2861 - 3s/epoch - 48ms/step
Epoch 195/200
68/68 - 3s - loss: 1.4245 - acc: 0.5778 - val_loss: 2.0528 - val_acc: 0.2861 - 3s/epoch - 40ms/step
Epoch 196/200
68/68 - 3s - loss: 1.4118 - acc: 0.5926 - val_loss: 2.0342 - val_acc: 0.3083 - 3s/epoch - 46ms/step
Epoch 197/200
68/68 - 3s - loss: 1.3895 - acc: 0.5926 - val_loss: 2.0361 - val_acc: 0.3083 - 3s/epoch - 47ms/step
Epoch 198/200
68/68 - 3s - loss: 1.4136 - acc: 0.5778 - val_loss: 2.0241 - val_acc: 0.3167 - 3s/epoch - 43ms/step
Epoch 199/200
68/68 - 3s - loss: 1.3973 - acc: 0.5750 - val_loss: 2.0255 - val_acc: 0.3222 - 3s/epoch - 46ms/step
Epoch 200/200
68/68 - 3s - loss: 1.3753 - acc: 0.5926 - val_loss: 2.0164 - val_acc: 0.3167 - 3s/epoch - 44ms/step
```

```
In [5]: 1 results = get_results(model_history,model,X_test,y_test, ref.labels.unique())  
2 results.create_plot(model_history)  
3 results.create_results(model)  
4 results.confusion_results(X_test, y_test, ref.labels.unique(), model)
```

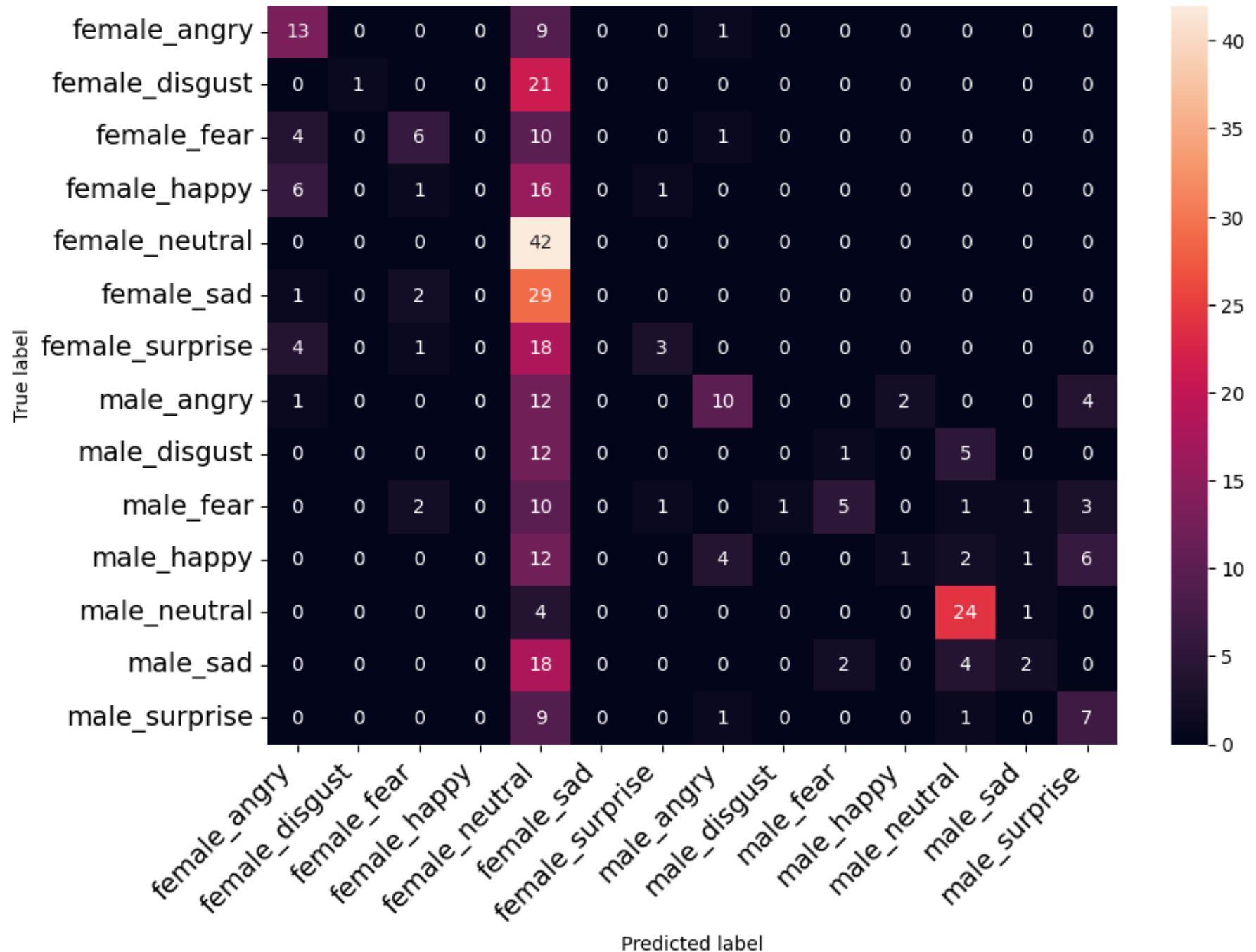



```
2022-10-19 16:33:16.235731: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_type GPU is enabled.
```

accuracy: 31.67%

```
2022-10-19 16:33:16.589391: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_type GPU is enabled.
```

23/23 - 0s - 281ms/epoch - 12ms/step



In []:

1

In []:

1