


```
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]:

Data Augmentation method

```
def speedNpitch(data):
    """
    Speed and Pitch Tuning.
    """
    # you can change low and high here
    length_change = np.random.uniform(low=0.8, high = 1)
    speed_fac = 1.2 / length_change # try changing 1.0 to 2.0 ... =D
    tmp = np.interp(np.arange(0,len(data)),speed_fac,np.arange(0,len(data)),data)
    minlen = min(data.shape[0], tmp.shape[0])
    data *= 0
    data[0:minlen] = tmp[0:minlen]
    return data
```

Extracting the MFCC feature as an image (Matrix format).

```
def prepare_data(df, n, aug, mfcc):
    X = np.empty(shape=(df.shape[0], n, 216, 1))
    input_length = sampling_rate * audio_duration

    cnt = 0
    for fname in tqdm(df.path):
        file_path = fname
        data, _ = librosa.load(file_path, sr=sampling_rate
                               ,res_type="kaiser_fast"
                               ,duration=2.5
                               ,offset=0.5
                               )

        # Random offset / Padding
        if len(data) > input_length:
            max_offset = len(data) - input_length
            offset = np.random.randint(max_offset)
            data = data[offset:(input_length+offset)]
        else:
            if input_length > len(data):
                max_offset = input_length - len(data)
                offset = np.random.randint(max_offset)
```

```

        else:
            offset = 0
            data = np.pad(data, (offset, int(input_length) - len(data) - offset), "constant")

    # Augmentation?
    if aug == 1:
        data = speedNpitch(data)

    # which feature?
    if mfcc == 1:
        # MFCC extraction
        MFCC = librosa.feature.mfcc(data, sr=sampling_rate, n_mfcc=n_mfcc)
        MFCC = np.expand_dims(MFCC, axis=-1)
        X[cnt,] = MFCC

    else:
        # Log-melspectrogram
        melspec = librosa.feature.melspectrogram(data, n_mels = n_melspec)
        logspec = librosa.amplitude_to_db(melspec)
        logspec = np.expand_dims(logspec, axis=-1)
        X[cnt,] = logspec

    cnt += 1

return X

```

Confusion matrix plot

```

: print_confusion_matrix(confusion_matrix, class_names, figsize = (10,7), fontsize=14):
'''Prints a confusion matrix, as returned by sklearn.metrics.confusion_matrix, as a heatmap.

Arguments
-----
confusion_matrix: numpy.ndarray
    The numpy.ndarray object returned from a call to sklearn.metrics.confusion_matrix.
    Similarly constructed ndarrays can also be used.
class_names: list
    An ordered list of class names, in the order they index the given confusion matrix.
figsize: tuple
    A 2-long tuple, the first value determining the horizontal size of the ouputted figure,

```

```

        the second determining the vertical size. Defaults to (10,7).
    fontsize: int
        Font size for axes labels. Defaults to 14.

Returns
-----
matplotlib.figure.Figure
    The resulting confusion matrix figure
    ...
df_cm = pd.DataFrame(
    confusion_matrix, index=class_names, columns=class_names,
)
fig = plt.figure(figsize=figsize)
try:
    heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
except ValueError:
    raise ValueError("Confusion matrix values must be integers.")

heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=fontsize)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', fontsize=fontsize)
plt.ylabel('True label')
plt.xlabel('Predicted label')

```

4. Create the 2D CNN model

```

def get_2d_conv_model(n):
    ''' Create a standard deep 2D convolutional neural network'''
    nclass = 14
    inp = Input(shape=(n,216,1)) #2D matrix of 30 MFCC bands by 216 audio length.
    x = Convolution2D(32, (4,10), padding="same")(inp)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)
    x = Dropout(rate=0.2)(x)

    x = Convolution2D(32, (4,10), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)

```

```

x = Dropout(rate=0.2)(x)

x = Convolution2D(32, (4,10), padding="same")(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = MaxPool2D()(x)
x = Dropout(rate=0.2)(x)

x = Convolution2D(32, (4,10), padding="same")(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = MaxPool2D()(x)
x = Dropout(rate=0.2)(x)

x = Flatten()(x)
x = Dense(64)(x)
x = Dropout(rate=0.2)(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Dropout(rate=0.2)(x)

out = Dense(nclass, activation=softmax)(x)
model = models.Model(inputs=inp, outputs=out)

opt = optimizers.Adam(0.0001)
    opt = keras.optimizers.RMSprop(lr=0.00001, decay=1e-6)
model.compile(optimizer=opt, loss=losses.categorical_crossentropy, metrics=['acc'])
model.summary()
return model

```

5. Other functions

```
def get_results:
```

```
'''
```

We're going to create a class (blueprint template) for generating the results based on the various model approaches. So instead of repeating the functions each time, we assign the results into an object with its associated variables depending on each combination:

- 1) MFCC with no augmentation
- 2) MFCC with augmentation
- 3) Logmelspec with no augmentation
- 4) Logmelspec with augmentation

```

'''

def __init__(self, model_history, model ,X_test, y_test, labels):
    self.model_history = model_history
    self.model = model
    self.X_test = X_test
    self.y_test = y_test
    self.labels = labels

def create_plot(self, model_history):
    '''Check the logloss of both train and validation, make sure they are close and have plateau'''
    plt.plot(model_history.history['loss'])
    plt.plot(model_history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()

def create_results(self, model):
    '''predict on test set and get accuracy results'''
    opt = optimizers.Adam(0.0001)
    model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
    score = model.evaluate(X_test, y_test, verbose=0)
    print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))

def confusion_results(self, X_test, y_test, labels, model):
    '''plot confusion matrix results'''
    preds = model.predict(X_test,
                           batch_size=16,
                           verbose=2)

    preds=preds.argmax(axis=1)
    preds = preds.astype(int).flatten()
    preds = (lb.inverse_transform((preds)))

    actual = y_test.argmax(axis=1)
    actual = actual.astype(int).flatten()
    actual = (lb.inverse_transform((actual)))

    classes = labels
    classes.sort()

```



```

c = confusion_matrix(actual, preds)
print_confusion_matrix(c, class_names = classes)

def accuracy_results_gender(self, X_test, y_test, labels, model):
    '''Print out the accuracy score and confusion matrix heat map of the Gender classification results'''

    preds = model.predict(X_test,
                           batch_size=16,
                           verbose=2)
    preds=preds.argmax(axis=1)
    preds = preds.astype(int).flatten()
    preds = (lb.inverse_transform((preds)))

    actual = y_test.argmax(axis=1)
    actual = actual.astype(int).flatten()
    actual = (lb.inverse_transform((actual)))

    # print(accuracy_score(actual, preds))

    actual = pd.DataFrame(actual).replace({'female_angry': 'female'
                                           , 'female_disgust': 'female'
                                           , 'female_fear': 'female'
                                           , 'female_happy': 'female'
                                           , 'female_sad': 'female'
                                           , 'female_surprise': 'female'
                                           , 'female_neutral': 'female'
                                           , 'male_angry': 'male'
                                           , 'male_fear': 'male'
                                           , 'male_happy': 'male'
                                           , 'male_sad': 'male'
                                           , 'male_surprise': 'male'
                                           , 'male_neutral': 'male'
                                           , 'male_disgust': 'male'
                                           })

    preds = pd.DataFrame(preds).replace({'female_angry': 'female'
                                           , 'female_disgust': 'female'
                                           , 'female_fear': 'female'
                                           , 'female_happy': 'female'
                                           , 'female_sad': 'female'
                                           , 'female_surprise': 'female'
                                           , 'female_neutral': 'female'
                                           , 'male_angry': 'male'

```

```
        , 'male_fear': 'male'  
        , 'male_happy': 'male'  
        , 'male_sad': 'male'  
        , 'male_surprise': 'male'  
        , 'male_neutral': 'male'  
        , 'male_disgust': 'male'  
    })  
  
    classes = actual.loc[:,0].unique()  
    classes.sort()  
  
    c = confusion_matrix(actual, preds)  
    print(accuracy_score(actual, preds))  
    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=100)

```

100%|██| 1440/1440 [00:26<00:00, 55.17it/s]

2022-10-13 20:20:41.797162: 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.

2022-10-13 20:20:41.797261: 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 1/100

```

2022-10-13 20:20:41.955673: W tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU frequency: 0 Hz
2022-10-13 20:20:42.227224: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_type GPU is enabled.
2022-10-13 20:20:43.912146: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_type GPU is enabled.

```

```

68/68 - 2s - loss: 2.9299 - acc: 0.0917 - val_loss: 2.6849 - val_acc: 0.0722 - 2s/epoch - 32ms/step
Epoch 2/100

```

```
68/68 - 2s - loss: 2.5982 - acc: 0.1694 - val_loss: 2.7021 - val_acc: 0.1056 - 2s/epoch - 22ms/step
Epoch 3/100
68/68 - 1s - loss: 2.4365 - acc: 0.2120 - val_loss: 2.6993 - val_acc: 0.1333 - 1s/epoch - 22ms/step
Epoch 4/100
68/68 - 1s - loss: 2.2987 - acc: 0.2472 - val_loss: 2.5969 - val_acc: 0.1472 - 1s/epoch - 21ms/step
Epoch 5/100
68/68 - 2s - loss: 2.2073 - acc: 0.2935 - val_loss: 2.5254 - val_acc: 0.1778 - 2s/epoch - 23ms/step
Epoch 6/100
68/68 - 1s - loss: 2.1621 - acc: 0.2806 - val_loss: 2.4337 - val_acc: 0.1917 - 1s/epoch - 21ms/step
Epoch 7/100
68/68 - 1s - loss: 2.1355 - acc: 0.3046 - val_loss: 2.4028 - val_acc: 0.1861 - 1s/epoch - 22ms/step
Epoch 8/100
68/68 - 1s - loss: 2.0645 - acc: 0.3194 - val_loss: 2.3888 - val_acc: 0.2000 - 1s/epoch - 21ms/step
Epoch 9/100
68/68 - 2s - loss: 2.0190 - acc: 0.3407 - val_loss: 2.3170 - val_acc: 0.2111 - 2s/epoch - 23ms/step
Epoch 10/100
68/68 - 2s - loss: 1.9535 - acc: 0.3787 - val_loss: 2.3167 - val_acc: 0.2167 - 2s/epoch - 22ms/step
Epoch 11/100
68/68 - 1s - loss: 1.9317 - acc: 0.3593 - val_loss: 2.3419 - val_acc: 0.2028 - 1s/epoch - 22ms/step
Epoch 12/100
68/68 - 1s - loss: 1.9153 - acc: 0.3778 - val_loss: 2.2898 - val_acc: 0.2333 - 1s/epoch - 22ms/step
Epoch 13/100
68/68 - 1s - loss: 1.8700 - acc: 0.4056 - val_loss: 2.3640 - val_acc: 0.1944 - 1s/epoch - 21ms/step
Epoch 14/100
68/68 - 2s - loss: 1.8445 - acc: 0.4019 - val_loss: 2.2749 - val_acc: 0.2139 - 2s/epoch - 23ms/step
Epoch 15/100
68/68 - 1s - loss: 1.7964 - acc: 0.4176 - val_loss: 2.2893 - val_acc: 0.2222 - 1s/epoch - 22ms/step
Epoch 16/100
68/68 - 2s - loss: 1.7409 - acc: 0.4472 - val_loss: 2.2279 - val_acc: 0.2250 - 2s/epoch - 22ms/step
Epoch 17/100
68/68 - 1s - loss: 1.6906 - acc: 0.4639 - val_loss: 2.2568 - val_acc: 0.2194 - 1s/epoch - 22ms/step
Epoch 18/100
68/68 - 1s - loss: 1.6462 - acc: 0.4815 - val_loss: 2.3203 - val_acc: 0.1778 - 1s/epoch - 22ms/step
Epoch 19/100
68/68 - 1s - loss: 1.6026 - acc: 0.4972 - val_loss: 2.1943 - val_acc: 0.2250 - 1s/epoch - 21ms/step
Epoch 20/100
68/68 - 2s - loss: 1.5365 - acc: 0.5444 - val_loss: 2.0985 - val_acc: 0.2417 - 2s/epoch - 23ms/step
Epoch 21/100
68/68 - 2s - loss: 1.5225 - acc: 0.5259 - val_loss: 2.1388 - val_acc: 0.2250 - 2s/epoch - 22ms/step
Epoch 22/100
68/68 - 1s - loss: 1.4858 - acc: 0.5259 - val_loss: 2.0687 - val_acc: 0.2639 - 1s/epoch - 22ms/step
Epoch 23/100
```

```
68/68 - 1s - loss: 1.4351 - acc: 0.5565 - val_loss: 1.9993 - val_acc: 0.3083 - 1s/epoch - 22ms/step
Epoch 24/100
68/68 - 2s - loss: 1.4318 - acc: 0.5463 - val_loss: 1.9885 - val_acc: 0.3111 - 2s/epoch - 22ms/step
Epoch 25/100
68/68 - 1s - loss: 1.3640 - acc: 0.5852 - val_loss: 1.8166 - val_acc: 0.4222 - 1s/epoch - 22ms/step
Epoch 26/100
68/68 - 1s - loss: 1.3252 - acc: 0.6204 - val_loss: 1.8157 - val_acc: 0.4250 - 1s/epoch - 22ms/step
Epoch 27/100
68/68 - 1s - loss: 1.3044 - acc: 0.6139 - val_loss: 1.8122 - val_acc: 0.4194 - 1s/epoch - 22ms/step
Epoch 28/100
68/68 - 1s - loss: 1.2736 - acc: 0.6185 - val_loss: 1.8593 - val_acc: 0.3722 - 1s/epoch - 22ms/step
Epoch 29/100
68/68 - 2s - loss: 1.2219 - acc: 0.6444 - val_loss: 1.6984 - val_acc: 0.5000 - 2s/epoch - 22ms/step
Epoch 30/100
68/68 - 1s - loss: 1.1864 - acc: 0.6491 - val_loss: 1.6776 - val_acc: 0.4972 - 1s/epoch - 22ms/step
Epoch 31/100
68/68 - 2s - loss: 1.1529 - acc: 0.6824 - val_loss: 1.7791 - val_acc: 0.4333 - 2s/epoch - 22ms/step
Epoch 32/100
68/68 - 1s - loss: 1.0889 - acc: 0.6944 - val_loss: 1.6619 - val_acc: 0.5028 - 1s/epoch - 22ms/step
Epoch 33/100
68/68 - 2s - loss: 1.0876 - acc: 0.6787 - val_loss: 1.6378 - val_acc: 0.5306 - 2s/epoch - 22ms/step
Epoch 34/100
68/68 - 1s - loss: 1.0415 - acc: 0.7111 - val_loss: 1.7055 - val_acc: 0.4806 - 1s/epoch - 22ms/step
Epoch 35/100
68/68 - 1s - loss: 1.0089 - acc: 0.7352 - val_loss: 1.5456 - val_acc: 0.5306 - 1s/epoch - 22ms/step
Epoch 36/100
68/68 - 1s - loss: 1.0109 - acc: 0.7185 - val_loss: 1.5382 - val_acc: 0.5333 - 1s/epoch - 21ms/step
Epoch 37/100
68/68 - 2s - loss: 0.9764 - acc: 0.7241 - val_loss: 1.5314 - val_acc: 0.5833 - 2s/epoch - 23ms/step
Epoch 38/100
68/68 - 1s - loss: 0.9570 - acc: 0.7407 - val_loss: 1.5567 - val_acc: 0.4806 - 1s/epoch - 22ms/step
Epoch 39/100
68/68 - 1s - loss: 0.9327 - acc: 0.7454 - val_loss: 1.5230 - val_acc: 0.5611 - 1s/epoch - 21ms/step
Epoch 40/100
68/68 - 1s - loss: 0.8961 - acc: 0.7750 - val_loss: 1.4779 - val_acc: 0.5333 - 1s/epoch - 22ms/step
Epoch 41/100
68/68 - 1s - loss: 0.8527 - acc: 0.7880 - val_loss: 1.4274 - val_acc: 0.5639 - 1s/epoch - 21ms/step
Epoch 42/100
68/68 - 1s - loss: 0.8333 - acc: 0.7944 - val_loss: 1.4451 - val_acc: 0.5722 - 1s/epoch - 21ms/step
Epoch 43/100
68/68 - 2s - loss: 0.8214 - acc: 0.8083 - val_loss: 1.4330 - val_acc: 0.5472 - 2s/epoch - 22ms/step
Epoch 44/100
```

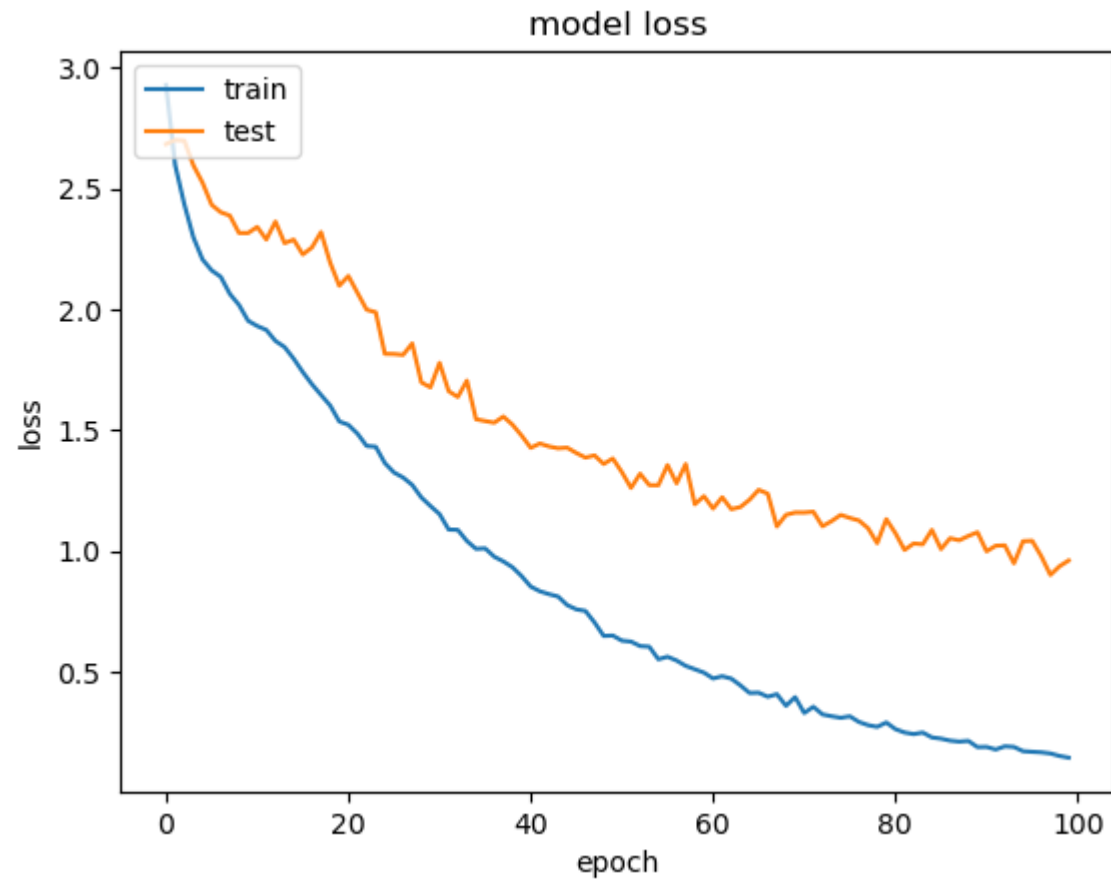
```
68/68 - 1s - loss: 0.8116 - acc: 0.8074 - val_loss: 1.4254 - val_acc: 0.5639 - 1s/epoch - 22ms/step
Epoch 45/100
68/68 - 1s - loss: 0.7760 - acc: 0.8046 - val_loss: 1.4278 - val_acc: 0.5361 - 1s/epoch - 22ms/step
Epoch 46/100
68/68 - 2s - loss: 0.7585 - acc: 0.8037 - val_loss: 1.4060 - val_acc: 0.5278 - 2s/epoch - 22ms/step
Epoch 47/100
68/68 - 1s - loss: 0.7517 - acc: 0.8148 - val_loss: 1.3868 - val_acc: 0.5861 - 1s/epoch - 22ms/step
Epoch 48/100
68/68 - 2s - loss: 0.7041 - acc: 0.8315 - val_loss: 1.3958 - val_acc: 0.5639 - 2s/epoch - 22ms/step
Epoch 49/100
68/68 - 1s - loss: 0.6485 - acc: 0.8556 - val_loss: 1.3605 - val_acc: 0.5833 - 1s/epoch - 21ms/step
Epoch 50/100
68/68 - 1s - loss: 0.6500 - acc: 0.8537 - val_loss: 1.3831 - val_acc: 0.5222 - 1s/epoch - 22ms/step
Epoch 51/100
68/68 - 2s - loss: 0.6288 - acc: 0.8602 - val_loss: 1.3264 - val_acc: 0.5556 - 2s/epoch - 22ms/step
Epoch 52/100
68/68 - 2s - loss: 0.6249 - acc: 0.8528 - val_loss: 1.2614 - val_acc: 0.5917 - 2s/epoch - 22ms/step
Epoch 53/100
68/68 - 1s - loss: 0.6064 - acc: 0.8574 - val_loss: 1.3200 - val_acc: 0.5972 - 1s/epoch - 21ms/step
Epoch 54/100
68/68 - 1s - loss: 0.6039 - acc: 0.8667 - val_loss: 1.2728 - val_acc: 0.5917 - 1s/epoch - 21ms/step
Epoch 55/100
68/68 - 1s - loss: 0.5517 - acc: 0.8778 - val_loss: 1.2725 - val_acc: 0.5889 - 1s/epoch - 22ms/step
Epoch 56/100
68/68 - 2s - loss: 0.5623 - acc: 0.8880 - val_loss: 1.3559 - val_acc: 0.5250 - 2s/epoch - 22ms/step
Epoch 57/100
68/68 - 1s - loss: 0.5465 - acc: 0.8806 - val_loss: 1.2803 - val_acc: 0.6222 - 1s/epoch - 21ms/step
Epoch 58/100
68/68 - 1s - loss: 0.5244 - acc: 0.8954 - val_loss: 1.3603 - val_acc: 0.5972 - 1s/epoch - 22ms/step
Epoch 59/100
68/68 - 2s - loss: 0.5103 - acc: 0.8991 - val_loss: 1.1936 - val_acc: 0.5972 - 2s/epoch - 23ms/step
Epoch 60/100
68/68 - 2s - loss: 0.4963 - acc: 0.8898 - val_loss: 1.2270 - val_acc: 0.6222 - 2s/epoch - 22ms/step
Epoch 61/100
68/68 - 1s - loss: 0.4722 - acc: 0.9120 - val_loss: 1.1758 - val_acc: 0.6111 - 1s/epoch - 21ms/step
Epoch 62/100
68/68 - 2s - loss: 0.4815 - acc: 0.9000 - val_loss: 1.2227 - val_acc: 0.6056 - 2s/epoch - 22ms/step
Epoch 63/100
68/68 - 1s - loss: 0.4716 - acc: 0.9009 - val_loss: 1.1732 - val_acc: 0.6444 - 1s/epoch - 21ms/step
Epoch 64/100
68/68 - 2s - loss: 0.4437 - acc: 0.9130 - val_loss: 1.1822 - val_acc: 0.6139 - 2s/epoch - 22ms/step
Epoch 65/100
```



```
68/68 - 1s - loss: 0.4122 - acc: 0.9259 - val_loss: 1.2124 - val_acc: 0.5639 - 1s/epoch - 22ms/step
Epoch 66/100
68/68 - 1s - loss: 0.4133 - acc: 0.9306 - val_loss: 1.2533 - val_acc: 0.5639 - 1s/epoch - 22ms/step
Epoch 67/100
68/68 - 1s - loss: 0.3966 - acc: 0.9333 - val_loss: 1.2373 - val_acc: 0.5806 - 1s/epoch - 21ms/step
Epoch 68/100
68/68 - 2s - loss: 0.4072 - acc: 0.9296 - val_loss: 1.1018 - val_acc: 0.6444 - 2s/epoch - 23ms/step
Epoch 69/100
68/68 - 1s - loss: 0.3591 - acc: 0.9417 - val_loss: 1.1498 - val_acc: 0.6194 - 1s/epoch - 22ms/step
Epoch 70/100
68/68 - 1s - loss: 0.3951 - acc: 0.9167 - val_loss: 1.1594 - val_acc: 0.6167 - 1s/epoch - 21ms/step
Epoch 71/100
68/68 - 1s - loss: 0.3292 - acc: 0.9491 - val_loss: 1.1593 - val_acc: 0.6056 - 1s/epoch - 21ms/step
Epoch 72/100
68/68 - 1s - loss: 0.3551 - acc: 0.9444 - val_loss: 1.1626 - val_acc: 0.6167 - 1s/epoch - 21ms/step
Epoch 73/100
68/68 - 1s - loss: 0.3234 - acc: 0.9444 - val_loss: 1.1026 - val_acc: 0.6528 - 1s/epoch - 22ms/step
Epoch 74/100
68/68 - 1s - loss: 0.3158 - acc: 0.9528 - val_loss: 1.1239 - val_acc: 0.6056 - 1s/epoch - 21ms/step
Epoch 75/100
68/68 - 2s - loss: 0.3094 - acc: 0.9491 - val_loss: 1.1496 - val_acc: 0.5833 - 2s/epoch - 23ms/step
Epoch 76/100
68/68 - 1s - loss: 0.3156 - acc: 0.9491 - val_loss: 1.1369 - val_acc: 0.6139 - 1s/epoch - 21ms/step
Epoch 77/100
68/68 - 1s - loss: 0.2921 - acc: 0.9519 - val_loss: 1.1263 - val_acc: 0.6389 - 1s/epoch - 22ms/step
Epoch 78/100
68/68 - 1s - loss: 0.2790 - acc: 0.9574 - val_loss: 1.0945 - val_acc: 0.6167 - 1s/epoch - 21ms/step
Epoch 79/100
68/68 - 1s - loss: 0.2719 - acc: 0.9565 - val_loss: 1.0325 - val_acc: 0.6500 - 1s/epoch - 22ms/step
Epoch 80/100
68/68 - 2s - loss: 0.2899 - acc: 0.9472 - val_loss: 1.1319 - val_acc: 0.5944 - 2s/epoch - 22ms/step
Epoch 81/100
68/68 - 1s - loss: 0.2625 - acc: 0.9648 - val_loss: 1.0738 - val_acc: 0.6528 - 1s/epoch - 22ms/step
Epoch 82/100
68/68 - 2s - loss: 0.2485 - acc: 0.9657 - val_loss: 1.0044 - val_acc: 0.6500 - 2s/epoch - 22ms/step
Epoch 83/100
68/68 - 1s - loss: 0.2419 - acc: 0.9713 - val_loss: 1.0310 - val_acc: 0.6528 - 1s/epoch - 21ms/step
Epoch 84/100
68/68 - 1s - loss: 0.2483 - acc: 0.9685 - val_loss: 1.0279 - val_acc: 0.6444 - 1s/epoch - 22ms/step
Epoch 85/100
68/68 - 1s - loss: 0.2281 - acc: 0.9741 - val_loss: 1.0873 - val_acc: 0.6278 - 1s/epoch - 22ms/step
Epoch 86/100
```

```
68/68 - 1s - loss: 0.2232 - acc: 0.9685 - val_loss: 1.0079 - val_acc: 0.6583 - 1s/epoch - 21ms/step
Epoch 87/100
68/68 - 1s - loss: 0.2148 - acc: 0.9741 - val_loss: 1.0530 - val_acc: 0.6528 - 1s/epoch - 22ms/step
Epoch 88/100
68/68 - 1s - loss: 0.2107 - acc: 0.9769 - val_loss: 1.0443 - val_acc: 0.6444 - 1s/epoch - 22ms/step
Epoch 89/100
68/68 - 1s - loss: 0.2137 - acc: 0.9722 - val_loss: 1.0623 - val_acc: 0.6361 - 1s/epoch - 21ms/step
Epoch 90/100
68/68 - 1s - loss: 0.1880 - acc: 0.9796 - val_loss: 1.0777 - val_acc: 0.6278 - 1s/epoch - 21ms/step
Epoch 91/100
68/68 - 1s - loss: 0.1888 - acc: 0.9796 - val_loss: 0.9986 - val_acc: 0.6639 - 1s/epoch - 21ms/step
Epoch 92/100
68/68 - 2s - loss: 0.1781 - acc: 0.9750 - val_loss: 1.0218 - val_acc: 0.6444 - 2s/epoch - 22ms/step
Epoch 93/100
68/68 - 1s - loss: 0.1913 - acc: 0.9694 - val_loss: 1.0233 - val_acc: 0.6417 - 1s/epoch - 22ms/step
Epoch 94/100
68/68 - 2s - loss: 0.1888 - acc: 0.9667 - val_loss: 0.9490 - val_acc: 0.6778 - 2s/epoch - 22ms/step
Epoch 95/100
68/68 - 1s - loss: 0.1710 - acc: 0.9778 - val_loss: 1.0406 - val_acc: 0.6417 - 1s/epoch - 21ms/step
Epoch 96/100
68/68 - 1s - loss: 0.1686 - acc: 0.9898 - val_loss: 1.0422 - val_acc: 0.6389 - 1s/epoch - 22ms/step
Epoch 97/100
68/68 - 1s - loss: 0.1664 - acc: 0.9824 - val_loss: 0.9783 - val_acc: 0.6528 - 1s/epoch - 22ms/step
Epoch 98/100
68/68 - 1s - loss: 0.1618 - acc: 0.9759 - val_loss: 0.9010 - val_acc: 0.6833 - 1s/epoch - 21ms/step
Epoch 99/100
68/68 - 2s - loss: 0.1513 - acc: 0.9815 - val_loss: 0.9368 - val_acc: 0.6750 - 2s/epoch - 22ms/step
Epoch 100/100
68/68 - 1s - loss: 0.1443 - acc: 0.9852 - val_loss: 0.9607 - val_acc: 0.6639 - 1s/epoch - 22ms/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)
```

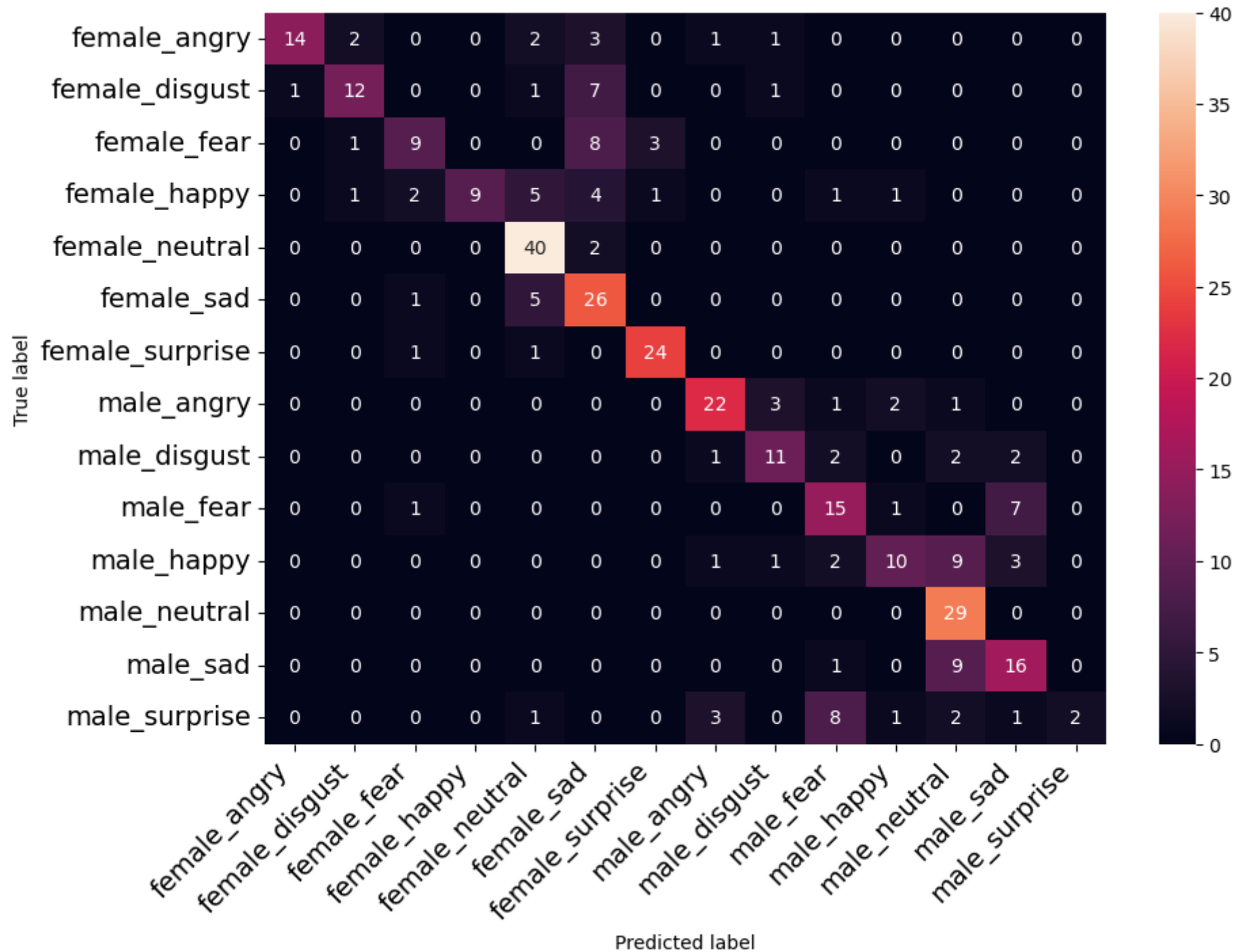


```
2022-10-13 20:23:11.055988: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_type GPU is enabled.
```

```
accuracy: 66.39%
```

```
23/23 - 0s - 189ms/epoch - 8ms/step
```

```
2022-10-13 20:23:11.319711: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_type GPU is enabled.
```



In []:

1