Lab1_Multi_Modal_Multi_Task

February 24, 2022

1 Lab: Multi-modal and Multi-task

by: Lawrence J. Klinkert and Jonas Moros

[41]:
```
# Code based on "F. Chollet, Deep learning with Python. 2021" work [2] and Dr.␣
↪Eric Larson's lecture notes/

import os
import pickle
import cv2
import mediapipe

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import sklearn.metrics as mt

import tensorflow as tf

from sklearn.utils import shuffle
from IPython.display import Image
from matplotlib import rcParams
from PIL import Image
from os import listdir
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from keras import layers
from keras import models
from keras.models import load_model
#from tensorflow.keras.utils import to_categorical

CWD = os.getcwd()
ORIGINAL_IMG_DIR = f"{CWD}\\Data\\RafD_original_data\\"
BASE_DIR = f"{CWD}\\Data\\RafD\\"
```

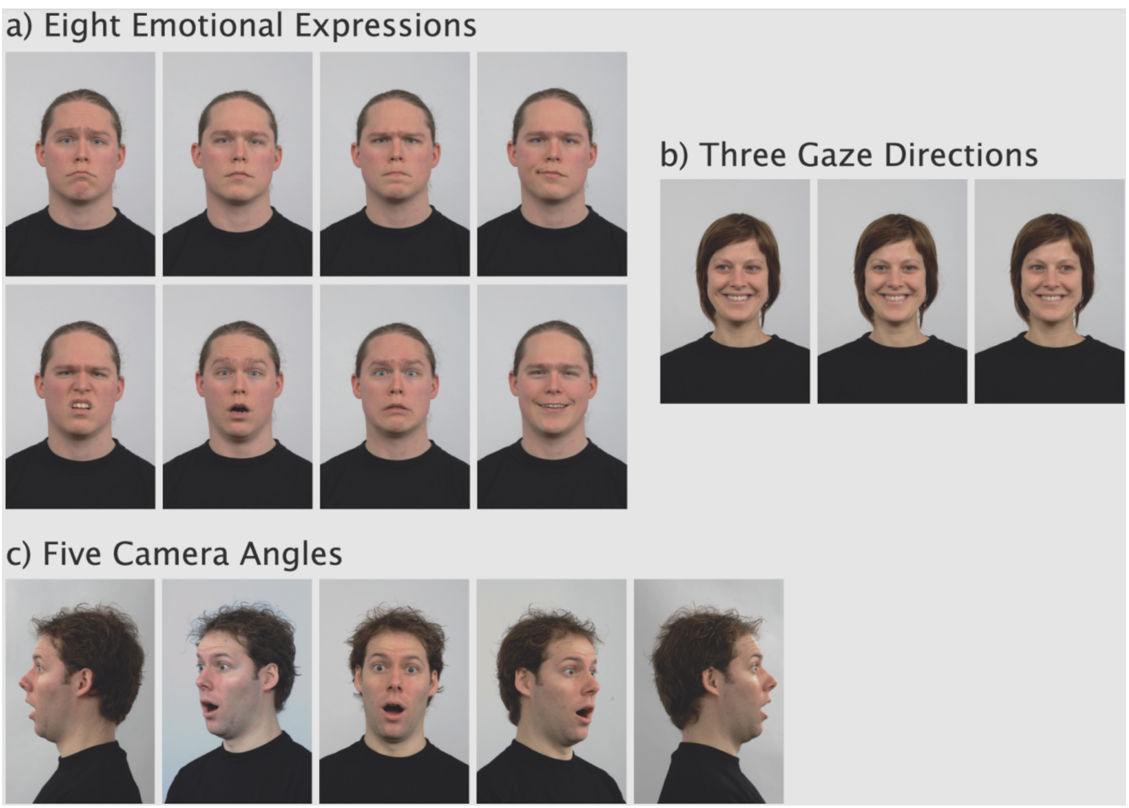
```
EMOTIONAL_CATEGORIES = ["angry", "contemptuous", "disgusted", "fearful",␣
↪"happy", "neutral", "sad", "surprised"]

MIN_Y = 128
MIN_X = 85
TOTAL_PIXLES = MIN_X * MIN_Y
TOTAL_PICS = 4824
TOTAL_LANDMARKS = 468
DF_IMAGE_DATA = None
```

1.1 General Overview of Dataset

The data set we will be using is the "Radboud Faces Database" provided by the Radboud University in Nijmegen, Netherlands [1]. The data set has 8040 pictures of actors exercising different emotions. We chose this data set because of the factorial design approach in gathering the images. There are 73 actors, 39 Caucasian Dutch adults, 10 Caucasian Dutch Children, and 23 Moroccans. Each actor practiced eight emotions: angry, contemptuous, disgusted, fearful, happy, neutral, sad, and surprised. When the actor was ready, five cameras took their picture simultaneously at different angles: 0 degrees, 45 degrees, 90 degrees, 135 degrees, and 180 degrees. Finally, for each emotion, the actor must perform it while gazing in a specific direction: front, right, or left. Below is an example image of each set from the paper [1]

[2]:
```
Image.open("Data/example1.PNG")
```

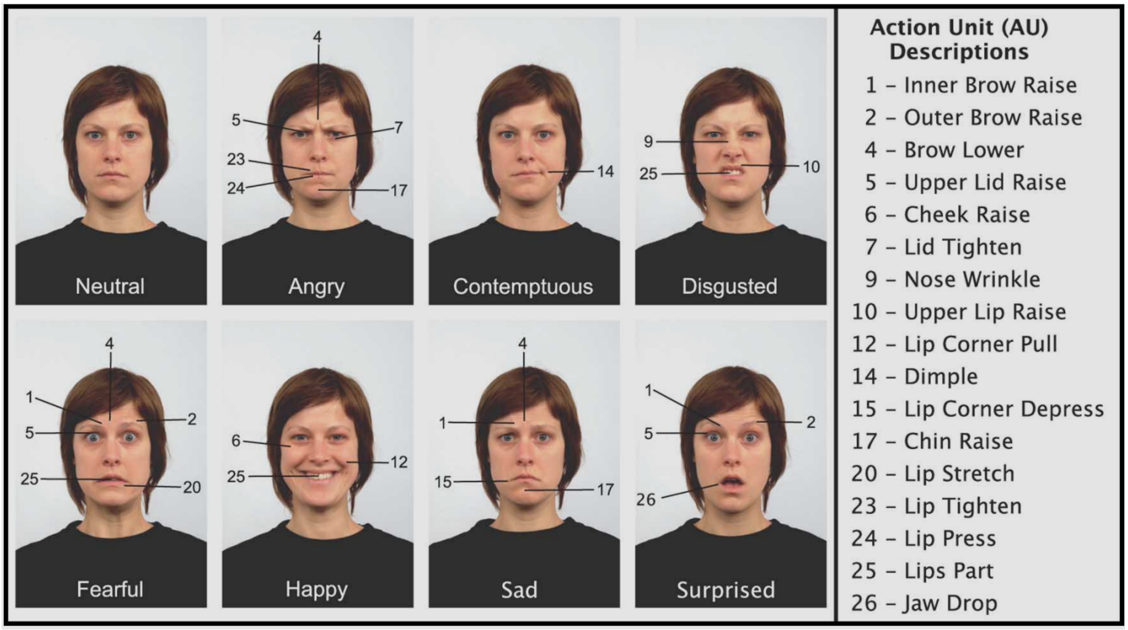
[2]:



Continuing from the Langner et al. paper, they mention the use of Action Units (AU). AUs are measurements used in the Facial Action Coding System (FACS). Alone, AU's will morph the face but combined allows for the expression of emotions. From fig 2, you can see that AUs are combined to create one of the eight different emotions.

[3]:
```
Image.open("Data/example2.PNG")
```

[3]:

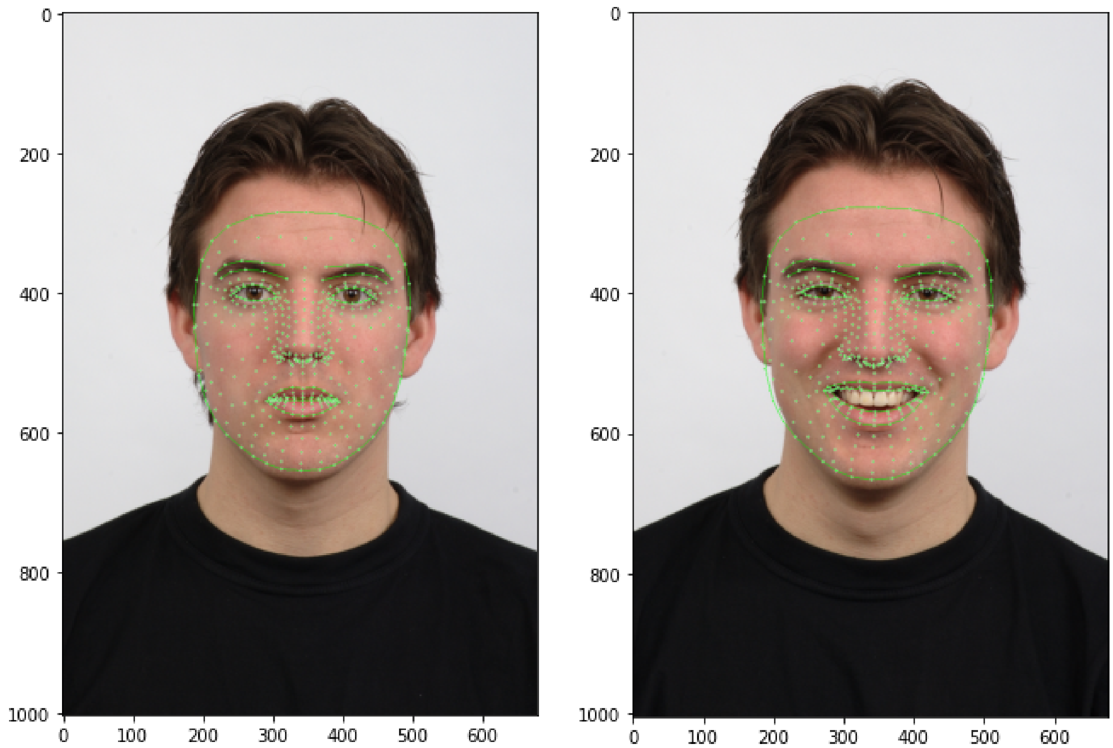


Since FACS is a propriety model, we could not find a dataset that measured each AU individually. However, we can use the same concept of FACS and attempt to create the measurement synthetically through Landmark Detection in Computer Vision. The idea is to use an off-the-shelf Facial Landmark Detection System, such as Google's MediaPipe in Python (https://google.github.io/mediapipe/getting_started/python.html), to mark key points on the face. Using this system on a neutral face will act as our origin. As the model emotes a face, we will mark the landmarks again and take the difference between the emote position vs the origin. We hypothesize that these emotional vectors, along with the images and positions, can help generalize the detection of expressive emotions.

[4]:
```
# figure size in inches optional
rcParams['figure.figsize'] = 11 ,8

EXAMPLE_IMG_DIR = f"{CWD}\\Data\\"

# read images
img_A = mpimg.imread(f"{EXAMPLE_IMG_DIR}ExNeutral.png")
img_B = mpimg.imread(f"{EXAMPLE_IMG_DIR}ExHappy.png")
```

```python
# display images
fig, ax = plt.subplots(1,2)
ax[0].imshow(img_A)
ax[1].imshow(img_B)
```

[4]: <matplotlib.image.AxesImage at 0x19ad6fc2da0>



1.A What is the classification task and what is the format of the feature data. Is this multi-task, multi-modal, or both? Explain.

We will construct a multimodal model that will use image data and euclidian coordinate data. The classification task is to classify the emotional expression from the picture. To construct the euclidian coordinate data, we will be constructing facial landmarks from images with MediaPipe. The module allows us to place a 468 point, three-dimensional, Face Mesh over the image. These will be our landmarks for neutral and emote images. By subtracting the neutral landmark points from the emotional landmark points, we can create a vector that describes the direction of the landmarks with respect to emotion.

For our initial design, we thought of using as input to model the neutral faces, the emotional faces, the neutral landmarks, the emotional landmarks, and the vector difference between the emotion vs. neutral landmarks. Regarding the task of classification, the neutral images and the neutral facial landmarks would not help classify an emote face, and these values would only change on an actor per actor instance. Moreover, having the same neutral face fed into the model may cause overfit of the neutral emote face classification. Thus, we decided that the first multimodal model will have the emote image and the vector difference between emoting and neutral facial landmark points. For the additional research question, we will look at a second multimodal modal that will input the emote image, vector difference, and emote facial landmark positions.

1.B. Who collected the data? Why? When?

The data was collected in 2010 by the Behavioral Science Institute of the Radboud University Nijmegen, located in the Netherlands. Their purpose for collecting was to formalize gathering emotional data over "in the wild" style photography for Affect research. Using "in the wild" style photos of individuals emoting preserves the genuineness of the expression at the cost of only having one instance from the actor. Finding essential features from the face becomes difficult, mainly when a default neutral position is not provided. Data augmentation could be a synthetic solution to construct more examples from "in the wild" data sets, but the transformation performed on the images are on two dimensions. Translating and rotating the image is not the problem, but rather it is moving the actor or the physical camera to provide additional examples. With the proposed factorial design, we can discover differences from neutral to the emote, change in angles, different gaze directions, all while using the same actor and emote. A validation process was performed to ensure that the genuineness was still apparent in the actor, and one can read about this process in their paper [1]. The results from training a ml model using the "Radboud Faces Database" may allow for knowledge transfer to another model whose task is to classify "in he wild" emote image datasets.

1.C What evaluation criteria will you be using and why?

We will be using accuracy and categorical accuracy. Accuracy and categorical accuracy will determine our frequent correct guesses for the emotion. Since we have a balanced classification of emotions, the ROC AUC score will help measure the model's ability to differentiate the classes. We use the Categorical Cross-Entropy Loss for the loss function since we have a multi-class classification of 8 different emotions.

2. How many tasks or modalities are there in the dataset and how do you define each task or modality? That is, explain if the task is within the same domain, cross domains, etc. If there are too many tasks or modalities to train the data reasonably, select a subset of the tasks for classification. For example, you might want to only train on 50 of the classification tasks.

There are two mediums, image, and numerical data, and two modalities the model will train on. The modalities are the emote image and the vector created by the difference between the emote and neutral landmark positions. We hypothesize that adding landmark directions will act as a heuristic for the model to narrow its classification selection of the expressed emotion.

1.2 Reading in Data

1.2.1 Loading Image Data

We would like to remove the profile shots images taken at 0 and 180 degrees since the entire face is not fully visible.

[146]:
```
#Removing side profile photos
images = [file for file in listdir(ORIGINAL_IMG_DIR) if file.endswith(('jpeg',␣
↪'png', 'jpg'))]
for image in images:
```

```
    path = ORIGINAL_IMG_DIR + image
    checker = image.split('_')
    if(checker[0] == 'Rafd000' or checker[0] == 'Rafd180'):
        print(path)
        os.remove(path)
```

[147]:

```
images = [file for file in listdir(ORIGINAL_IMG_DIR) if file.endswith(('jpeg',␣
 ↪'png', 'jpg'))]

l_names = []
l_rotation = []
l_model = []
l_type = []
l_sex = []
l_emotion = []
l_gaze = []

for image in images:
    split_string = image.split("_")

    name = image
    rotation = split_string[0].split("Rafd")[1]
    model = split_string[1]
    type = split_string[2]
    sex = split_string[3]
    emotion = split_string[4]
    gaze = split_string[5].split(".jpg")[0]

    l_names.append(name)
    l_rotation.append(rotation)
    l_model.append(model)
    l_type.append(type)
    l_sex.append(sex)
    l_emotion.append(emotion)
    l_gaze.append(gaze)

df_image_data = pd.DataFrame(data = list(zip(l_names, l_rotation, l_model,␣
 ↪l_type, l_sex, l_emotion, l_gaze)),
                         columns =['Name', 'Rotation', 'Model', 'Type', 'Sex',␣
 ↪'Emotion', 'Gaze'])

df_image_data = df_image_data.astype({'Name': 'object'})
df_image_data = df_image_data.astype({'Rotation': 'uint8'})
df_image_data = df_image_data.astype({'Model': 'uint8'})
df_image_data = df_image_data.astype({'Type': 'category'})
df_image_data = df_image_data.astype({'Sex': 'category'})
df_image_data = df_image_data.astype({'Emotion': 'category'})
```

```
df_image_data = df_image_data.astype({'Gaze': 'category'})
df_image_data
```

[147]:

```
                                             Name  Rotation  Model  \
0         Rafd045_01_Caucasian_female_angry_frontal.jpg        45      1
1            Rafd045_01_Caucasian_female_angry_left.jpg        45      1
2           Rafd045_01_Caucasian_female_angry_right.jpg        45      1
3     Rafd045_01_Caucasian_female_contemptuous_front…        45      1
4     Rafd045_01_Caucasian_female_contemptuous_left.jpg        45      1
…                                                   …       …      …
4819              Rafd135_73_Moroccan_male_sad_left.jpg       135     73
4820             Rafd135_73_Moroccan_male_sad_right.jpg       135     73
4821      Rafd135_73_Moroccan_male_surprised_frontal.jpg       135     73
4822         Rafd135_73_Moroccan_male_surprised_left.jpg       135     73
4823        Rafd135_73_Moroccan_male_surprised_right.jpg       135     73

           Type     Sex       Emotion     Gaze
0     Caucasian  female         angry  frontal
1     Caucasian  female         angry     left
2     Caucasian  female         angry    right
3     Caucasian  female  contemptuous  frontal
4     Caucasian  female  contemptuous     left
…           …      …             …       …
4819   Moroccan    male           sad     left
4820   Moroccan    male           sad    right
4821   Moroccan    male     surprised  frontal
4822   Moroccan    male     surprised     left
4823   Moroccan    male     surprised    right

[4824 rows x 7 columns]
```

1.2.2 Facial Landmark Coordinate Generation

[148]:

```
# Code based on the tutorial provided at https://techtutorialsx.com/2021/05/19/
↪mediapipe-face-landmarks-estimation/

# parallel array to dictionary image names
l_emote_landmark_data = list()

#drawingModule = mediapipe.solutions.drawing_utils
faceModule = mediapipe.solutions.face_mesh

#circleDrawingSpec = drawingModule.DrawingSpec(thickness=1, circle_radius=1,␣
↪color=(0,255,0))
#lineDrawingSpec = drawingModule.DrawingSpec(thickness=1, color=(0,255,0))

with faceModule.FaceMesh(static_image_mode=True) as face:
```

```
    image_list = df_image_data['Name'].to_list()
    image_number = 0
    for image_name in image_list:
        image_dir = os.path.join(ORIGINAL_IMG_DIR, image_name)
        image = cv2.imread(image_dir)

        results = face.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

        if results.multi_face_landmarks != None:
            for faceLandmarks in results.multi_face_landmarks:
                landmakrs = list()
                #drawingModule.draw_landmarks(image, faceLandmarks, faceModule.
↪FACEMESH_CONTOURS, circleDrawingSpec, lineDrawingSpec)
                #drawingModule.plot_landmarks(faceLandmarks, faceModule.
↪FACEMESH_CONTOURS, circleDrawingSpec, lineDrawingSpec)
                for landmakr in faceLandmarks.landmark:
                    coordinates = list()
                    coordinates.append(landmakr.x)
                    coordinates.append(landmakr.y)
                    coordinates.append(landmakr.z)
                    landmakrs.append(coordinates)
                    #print(f"(x: {landmakr.x}, y: {landmakr.y}, z: {landmakr.
↪z})")
                    #landmark_number += 1
                l_emote_landmark_data.append(landmakrs)
        image_number += 1


        #cv2.imshow('Test image', image)
        #cv2.waitKey(0)
    #cv2.destroyAllWindows()
```

[149]:
```
np_emote_landmark_data = np.asarray(l_emote_landmark_data)
np_emote_landmark_data.shape
```

[149]: (4824, 468, 3)

[150]:
```
np_emote_landmark_data[4000]
```

[150]:
```
array([[ 0.26492968,  0.53654552, -0.03691846],
       [ 0.2286239 ,  0.4898462 , -0.07798729],
       [ 0.26675886,  0.50331444, -0.03861171],
       …,
       [ 0.3251555 ,  0.41419742, -0.04281038],
       [ 0.44198859,  0.39975575, -0.09534382],
       [ 0.45592448,  0.39383322, -0.10329556]])
```

We take all the landmarks name the columns according to the landmark and put them into a dataframe.

[151]:
```python
col_name = list()

for num in range(0, TOTAL_LANDMARKS):
    col_name.append(f'L{num}')

df_emote_landmark_data = pd.DataFrame(l_emote_landmark_data, columns=col_name)
#df_landmark_data.insert(loc=0, column="Name", value=df_image_data.Name.
↪to_numpy())
df_emote_landmark_data
```

[151]:
```
                                                L0  \
0     [0.7945298552513123, 0.5117199420928955, -0.02…
1     [0.7952767610549927, 0.5149459838867188, -0.02…
2     [0.7934446334838867, 0.5115123987197876, -0.02…
3     [0.7885480523109436, 0.5205190181732178, -0.03…
4     [0.7951141595840454, 0.518377959728241, -0.029…
…                                                 …
4819  [0.2691039443016052, 0.527969241142273, -0.035…
4820  [0.2636050581932068, 0.5235021114349365, -0.04…
4821  [0.2543388307094574, 0.5253018140792847, -0.04…
4822  [0.2622019052505493, 0.541678786277771, -0.044…
4823  [0.2613617181777954, 0.5321192145347595, -0.04…

                                                L1  \
0     [0.8229753971099854, 0.4742351770401001, -0.05…
1     [0.8244158625602722, 0.4790429472923279, -0.06…
2     [0.8238111138343811, 0.4740161895751953, -0.05…
3     [0.81507408618927, 0.4807978570461273, -0.0599…
4     [0.8203902840614319, 0.47946441173553467, -0.0…
…                                                 …
4819  [0.23828883469104767, 0.48936066031455994, -0…
4820  [0.23722884058952332, 0.481378972530365, -0.07…
4821  [0.2236580103635788, 0.47856947779655457, -0.0…
4822  [0.22616463899612427, 0.49609479308128357, -0…
4823  [0.23180657625198364, 0.4863617420196533, -0.0…

                                                L2  \
0     [0.7881913781166077, 0.487883061170578, -0.025…
1     [0.7918961644172668, 0.49116250872612, -0.0291…
2     [0.7878001928329468, 0.4874943494796753, -0.02…
3     [0.7830095291137695, 0.4944821000099182, -0.02…
4     [0.7882686257362366, 0.49327608942985535, -0.0…
…                                                 …
4819  [0.2717481255531311, 0.5001242756843567, -0.03…
4820  [0.2678363025188446, 0.4944700300693512, -0.03…
```

```
4821  [0.2591533362865448, 0.4939173758029938, -0.03…
4822  [0.2640763521194458, 0.5108305811882019, -0.03…
4823  [0.2658877968788147, 0.5020670294761658, -0.03…

                                                 L3  \
0     [0.7915933728218079, 0.43765753507614136, -0.0…
1     [0.7928334474563599, 0.4421041011810303, -0.06…
2     [0.7914088368415833, 0.43719246983528137, -0.0…
3     [0.7844383716583252, 0.44387000799179077, -0.0…
4     [0.7874796390533447, 0.44342416524887085, -0.0…
…                                                 …
4819  [0.2509370446205139, 0.4483848810195923, -0.03…
4820  [0.2505646347999573, 0.4411831200122833, -0.03…
4821  [0.24137523770332336, 0.4374336898326874, -0.0…
4822  [0.2417217493057251, 0.4553801715373993, -0.02…
4823  [0.24872912466526031, 0.4462577700614929, -0.0…

                                                 L4  \
0     [0.825316846370697, 0.4622188210487366, -0.061…
1     [0.82639479637146, 0.4672662317752838, -0.0652…
2     [0.826101541519165, 0.4619814157485962, -0.060…
3     [0.817435085773468, 0.4690294563770294, -0.064…
4     [0.8220833539962769, 0.4678357243537903, -0.06…
…                                                 …
4819  [0.2361718714237213, 0.47699278593063354, -0.0…
4820  [0.2359379529953003, 0.46866533160209656, -0.0…
4821  [0.22231851518154144, 0.4651586413383484, -0.0…
4822  [0.2235712707042694, 0.4826546609401703, -0.06…
4823  [0.2304835021495819, 0.4728730320930481, -0.06…

                                                 L5  \
0     [0.8153240084648132, 0.44736015796661377, -0.0…
1     [0.8166989684104919, 0.4522327184677124, -0.06…
2     [0.8155953884124756, 0.4470082223415375, -0.05…
3     [0.8081496953964233, 0.45434316992759705, -0.0…
4     [0.8118000030517578, 0.45346856117248535, -0.0…
…                                                 …
4819  [0.24592439830303192, 0.46074551343917847, -0…
4820  [0.2462155520915985, 0.4527661204338074, -0.06…
4821  [0.23463837802410126, 0.4487839937210083, -0.0…
4822  [0.23487645387649536, 0.46612367033958435, -0…
4823  [0.24247553944587708, 0.45673003792762756, -0…

                                                 L6  \
0     [0.7809969186782837, 0.41016650199890137, -0.0…
1     [0.7830389738082886, 0.4150044620037079, -0.03…
2     [0.7797267436981201, 0.40960824489593506, -0.0…
```

```
3     [0.7753925919532776, 0.4165523648262024, -0.03…
4     [0.7764525413513184, 0.4163399636745453, -0.03…
…                                                 …
4819  [0.28047800064086914, 0.4199253022670746, -0.0…
4820  [0.28110945224761963, 0.4136214852333069, -0.0…
4821  [0.27573293447494507, 0.4084271788597107, -0.0…
4822  [0.27312105894088745, 0.42564886808395386, -0…
4823  [0.28179818391799927, 0.41768157482147217, -0…

                                                  L7  \
0     [0.6435821056365967, 0.41355952620506287, -0.0…
1     [0.6449966430664062, 0.41417479515075684, -0.0…
2     [0.6437345147132874, 0.4121924638748169, -0.09…
3     [0.6334807276725769, 0.4135039150714874, -0.08…
4     [0.635176420211792, 0.4155900478363037, -0.083…
…                                                 …
4819  [0.2526087164878845, 0.410665899515152, 0.1089…
4820  [0.24602967500686646, 0.4081208109855652, 0.11…
4821  [0.24683119356632233, 0.4073745608329773, 0.12…
4822  [0.2526901364326477, 0.43008849024772644, 0.12…
4823  [0.2547666132450104, 0.4225175976753235, 0.124…

                                                  L8  \
0     [0.7724859714508057, 0.38355162739753723, -0.0…
1     [0.7738161683082581, 0.3898659348487854, -0.03…
2     [0.7707639932632446, 0.3835305869579315, -0.03…
3     [0.7681708335876465, 0.38337746262550354, -0.0…
4     [0.7675599455833435, 0.38375574350357056, -0.0…
…                                                 …
4819  [0.28806358575820923, 0.3922383487224579, -0.0…
4820  [0.2898111343383789, 0.3850422501564026, -0.01…
4821  [0.2859981954097748, 0.3741373121738434, -0.01…
4822  [0.280168354511261, 0.39224502444267273, -0.00…
4823  [0.29135459661483765, 0.3844364881515503, -0.0…

                                                  L9  …  \
0     [0.775367021560669, 0.3683893382549286, -0.039…  …
1     [0.7762071490287781, 0.3751910924911499, -0.03…  …
2     [0.7736449837684631, 0.3685104548931122, -0.04…  …
3     [0.7718051075935364, 0.3670552372932434, -0.02…  …
4     [0.7704975605010986, 0.3675183653831482, -0.02…  …
…                                                 …  …
4819  [0.2844194769859314, 0.3772333264350891, -0.02…  …
4820  [0.28711551427841187, 0.36932265758514404, -0…  …
4821  [0.2833264470100403, 0.35691291093826294, -0.0…  …
4822  [0.27556610107421875, 0.3752201199531555, -0.0…  …
4823  [0.28867441415786743, 0.3673596978187561, -0.0…  …
```

```
                                                   L458  \
0     [0.8150545954704285, 0.47662651538848877, -0.0…
1     [0.818798840045929, 0.48138025403022766, -0.03…
2     [0.8153846263885498, 0.47624289989471436, -0.0…
3     [0.8084297180175781, 0.48410964012145996, -0.0…
4     [0.8136093616485596, 0.48264575004577637, -0.0…
…                                                 …
4819  [0.26870566606521606, 0.493918776512146, -0.06…
4820  [0.2665920555591583, 0.4869658946990967, -0.06…
4821  [0.2551984488964081, 0.4846409559249878, -0.06…
4822  [0.25783324241638184, 0.5011027455329895, -0.0…
4823  [0.26229313015937805, 0.4919675886631012, -0.0…

                                                   L459  \
0     [0.8211409449577332, 0.4718768000602722, -0.02…
1     [0.8248541355133057, 0.47684767842292786, -0.0…
2     [0.8214805126190186, 0.4715108275413513, -0.02…
3     [0.814486026763916, 0.4794779121875763, -0.030…
4     [0.8193590641021729, 0.47794482111930847, -0.0…
…                                                 …
4819  [0.2751959562301636, 0.49045610427856445, -0.0…
4820  [0.27366262674331665, 0.48332253098487854, -0…
4821  [0.2622184157371521, 0.4801879823207855, -0.07…
4822  [0.2638425827026367, 0.4961767792701721, -0.07…
4823  [0.26903867721557617, 0.4871620237827301, -0.0…

                                                   L460  \
0     [0.803556501865387, 0.47928786277770996, 0.007…
1     [0.8097369074821472, 0.48284757137298584, 0.00…
2     [0.8029727935791016, 0.47864270210266113, 0.00…
3     [0.7971327900886536, 0.48729199171066284, 0.00…
4     [0.8019800186157227, 0.4855445921421051, 0.004…
…                                                 …
4819  [0.3207301199436188, 0.4960290491580963, -0.04…
4820  [0.3168622851371765, 0.4912410378456116, -0.05…
4821  [0.3090481162071228, 0.4897628724575043, -0.05…
4822  [0.3118765354156494, 0.5040530562400818, -0.05…
4823  [0.3144943416118622, 0.49645280838012695, -0.0…

                                                   L461  \
0     [0.8173994421958923, 0.47817903757095337, -0.0…
1     [0.8206002712249756, 0.4829029142856598, -0.04…
2     [0.817841112613678, 0.47785401344299316, -0.03…
3     [0.8105849027633667, 0.485530823469162, -0.043…
4     [0.8158515691757202, 0.4840426445007324, -0.04…
…                                                 …
```

```
4819  [0.26238512992858887, 0.49470850825309753, -0…
4820  [0.26040810346603394, 0.4875892996788025, -0.0…
4821  [0.24846360087394714, 0.4853466749191284, -0.0…
4822  [0.25114113092422485, 0.5020467638969421, -0.0…
4823  [0.25561147928237915, 0.49282771348953247, -0…

                                                L462  \
0     [0.805298388004303, 0.4814191460609436, -0.026…
1     [0.8095834255218506, 0.48563382029533386, -0.0…
2     [0.8055053949356079, 0.4810420870780945, -0.02…
3     [0.7990750074386597, 0.48835498094558716, -0.0…
4     [0.8043736219406128, 0.48698854446411133, -0.0…
…                                              …
4819  [0.2716958820819855, 0.4968680739402771, -0.04…
4820  [0.268843412399292, 0.4903632402420044, -0.052…
4821  [0.2581833004951477, 0.48890623450279236, -0.0…
4822  [0.2617035508155823, 0.5055043697357178, -0.05…
4823  [0.26518386602401733, 0.4965592622756958, -0.0…

                                                L463  \
0     [0.7697353363037109, 0.4029555022716522, 0.036…
1     [0.7723058462142944, 0.40851128101348877, 0.03…
2     [0.7663955688476562, 0.40201330184936523, 0.03…
3     [0.7692477107048035, 0.4115055501461029, 0.040…
4     [0.7690775394439697, 0.41139286756515503, 0.04…
…                                              …
4819  [0.35088375210762024, 0.4119616150856018, -0.0…
4820  [0.3512524962425232, 0.40868017077445984, -0.0…
4821  [0.3504284620285034, 0.40170755982398987, -0.0…
4822  [0.3449722230434418, 0.41623860597610474, -0.0…
4823  [0.35402166843414307, 0.41043904423713684, -0…

                                                L464  \
0     [0.7691131830215454, 0.4069206416606903, 0.020…
1     [0.7720801830291748, 0.4120901823043823, 0.021…
2     [0.7661666870117188, 0.406033456325531, 0.0197…
3     [0.7681577205657959, 0.4147629141807556, 0.024…
4     [0.7683023810386658, 0.41457295417785645, 0.02…
…                                              …
4819  [0.3374091684818268, 0.41612011194229126, -0.0…
4820  [0.33770161867141724, 0.41222429275512695, -0…
4821  [0.33614829182624817, 0.4059057831764221, -0.0…
4822  [0.33162790536880493, 0.42084765434265137, -0…
4823  [0.3403295576572418, 0.4146411716938019, -0.01…

                                                L465  \
0     [0.7745243310928345, 0.40966635942459106, 0.00…
```

```
1     [0.777540922164917, 0.41470077633857727, 0.008…
2     [0.7720775008201599, 0.40884876251220703, 0.00…
3     [0.7725656032562256, 0.4172639846801758, 0.013…
4     [0.7730149030685425, 0.4169284403324127, 0.013…
…                                                 …
4819  [0.32501763105392456, 0.4193998873233795, -0.0…
4820  [0.32527199387550354, 0.414747029542923, -0.02…
4821  [0.3226658701896675, 0.4086346924304962, -0.0…
4822  [0.3185201585292816, 0.4238968789577484, -0.02…
4823  [0.3272045850753784, 0.4172436594963074, -0.02…

                                                  L466  \
0     [0.7983791828155518, 0.3923327326774597, 0.102…
1     [0.804557204246521, 0.39925962686538696, 0.099…
2     [0.7931888103485107, 0.3920886218547821, 0.102…
3     [0.8099749088287354, 0.40508466958999634, 0.10…
4     [0.8078792095184326, 0.4040661156177521, 0.105…
…                                                 …
4819  [0.42765480279922485, 0.4043304920196533, -0.0…
4820  [0.42977219820022583, 0.4018123745918274, -0.0…
4821  [0.43376094102859497, 0.3900863528251648, -0.0…
4822  [0.4220271110534668, 0.4011582136154175, -0.07…
4823  [0.43294066190719604, 0.3979336619377136, -0.0…

                                                  L467
0     [0.7992184162139893, 0.3901614844799042, 0.109…
1     [0.8056724667549133, 0.39716973900794983, 0.10…
2     [0.7939038276672363, 0.3899647891521454, 0.109…
3     [0.8111420273780823, 0.4006441831588745, 0.112…
4     [0.8087610006332397, 0.3996562957763672, 0.112…
…                                                 …
4819  [0.4395391047000885, 0.39892685413360596, -0.0…
4820  [0.4422483742237091, 0.39608895778656006, -0.0…
4821  [0.4472726285457611, 0.3829534351825714, -0.08…
4822  [0.4349961578845978, 0.39287227392196655, -0.0…
4823  [0.4466977119445801, 0.3898908495903015, -0.08…

[4824 rows x 468 columns]
```

This dataframe is for book keeping. This is used inorder for us to take the difference from the emotional face and the neutral face.

[152]:
```
df_comparison = pd.DataFrame(df_image_data.Name.to_numpy(), columns=["Emote"])

nuetral_list = list()
for index, row in df_comparison.iterrows():
    name = row[0].removesuffix('.jpg')
```

```
    split_name = name.split('_')
    split_name[0] = "Rafd090"
    split_name[4] = "neutral"
    split_name[5] = "frontal"
    s = '_'
    s = s.join(split_name)
    s = s + ".jpg"
    nuetral_list.append(s)
nuetral_list

df_comparison = df_comparison.assign(Neutral = nuetral_list)
df_comparison
```

[152]:

```
                                                Emote  \
0         Rafd045_01_Caucasian_female_angry_frontal.jpg
1            Rafd045_01_Caucasian_female_angry_left.jpg
2           Rafd045_01_Caucasian_female_angry_right.jpg
3     Rafd045_01_Caucasian_female_contemptuous_front…
4     Rafd045_01_Caucasian_female_contemptuous_left.jpg
…                                                   …
4819              Rafd135_73_Moroccan_male_sad_left.jpg
4820             Rafd135_73_Moroccan_male_sad_right.jpg
4821     Rafd135_73_Moroccan_male_surprised_frontal.jpg
4822        Rafd135_73_Moroccan_male_surprised_left.jpg
4823       Rafd135_73_Moroccan_male_surprised_right.jpg

                                              Neutral
0     Rafd090_01_Caucasian_female_neutral_frontal.jpg
1     Rafd090_01_Caucasian_female_neutral_frontal.jpg
2     Rafd090_01_Caucasian_female_neutral_frontal.jpg
3     Rafd090_01_Caucasian_female_neutral_frontal.jpg
4     Rafd090_01_Caucasian_female_neutral_frontal.jpg
…                                                 …
4819     Rafd090_73_Moroccan_male_neutral_frontal.jpg
4820     Rafd090_73_Moroccan_male_neutral_frontal.jpg
4821     Rafd090_73_Moroccan_male_neutral_frontal.jpg
4822     Rafd090_73_Moroccan_male_neutral_frontal.jpg
4823     Rafd090_73_Moroccan_male_neutral_frontal.jpg

[4824 rows x 2 columns]
```

[153]:

```
l_landmarkDiff_data = list()

for index, row in df_comparison.iterrows():
    emote_coord = df_image_data.loc[df_image_data['Name'] == row.values[0]]
    neutral_coord = df_image_data.loc[df_image_data['Name'] == row.values[1]]
```

```
    landmakr_vector = list()
    for idx in range(0, TOTAL_LANDMARKS):
        vecotr = list()
        emot_x = l_emote_landmark_data[emote_coord.index[0]][idx][0]
        emot_y = l_emote_landmark_data[emote_coord.index[0]][idx][1]
        emot_z = l_emote_landmark_data[emote_coord.index[0]][idx][2]
        neutral_x = l_emote_landmark_data[neutral_coord.index[0]][idx][0]
        neutral_y = l_emote_landmark_data[neutral_coord.index[0]][idx][1]
        neutral_z = l_emote_landmark_data[neutral_coord.index[0]][idx][2]

        vecotr.append(emot_x - neutral_x)
        vecotr.append(emot_y - neutral_y)
        vecotr.append(emot_z - neutral_z)

        denominator = np.sqrt(vecotr[0]*vecotr[0] + vecotr[1]*vecotr[1] +
 ↪vecotr[2]*vecotr[2])

        vecotr[0] = vecotr[0] / denominator
        vecotr[1] = vecotr[1] / denominator
        vecotr[2] = vecotr[2] / denominator

        if vecotr[0] == np.nan:
            vecotr[0] = 0.0

        if vecotr[1] == np.nan:
            vecotr[1] = 0.0

        if vecotr[2] == np.nan:
            vecotr[2] = 0.0

        landmakr_vector.append(vecotr)
    l_landmarkDiff_data.append(landmakr_vector)
```

```
C:\Users\JAKEKL~1\AppData\Local\Temp/ipykernel_65644/2645951956.py:23:
RuntimeWarning: invalid value encountered in double_scalars
  vecotr[0] = vecotr[0] / denominator
C:\Users\JAKEKL~1\AppData\Local\Temp/ipykernel_65644/2645951956.py:24:
RuntimeWarning: invalid value encountered in double_scalars
  vecotr[1] = vecotr[1] / denominator
C:\Users\JAKEKL~1\AppData\Local\Temp/ipykernel_65644/2645951956.py:25:
RuntimeWarning: invalid value encountered in double_scalars
  vecotr[2] = vecotr[2] / denominator
```

[154]:
```
np_landmarkDiff_data = np.array(l_landmarkDiff_data)
np_landmarkDiff_data.shape
```

[154]:
```
(4824, 468, 3)
```

```
[155]: holder = []
       for x in range(0, TOTAL_PICS):
           temp = np.hstack(np_landmarkDiff_data[x])
           holder.append(temp)
```

```
[156]: col_name = list()

       for num in range(0, TOTAL_LANDMARKS):
           col_name.append(f'DeltaL{num}')

       df_landmarkDiff_data = pd.DataFrame(l_landmarkDiff_data, columns=col_name)
       df_landmark_data = df_emote_landmark_data.join(df_landmarkDiff_data)
```

```
[157]: image_list
       path = './Data/RafD_original_data/*'
       import glob

       def read_data_flatten():
           files = glob.glob(path)
           img_list = []
           for name in files:
               img = Image.open(name)
               img = img.resize((MIN_X,MIN_Y))
               data = np.asarray(img)
               data = data.flatten()
               data = data/255.0
               img_list.append(data)
           return img_list
       flatten_img_list = read_data_flatten()
```

```
[158]: new_flatten_img_list = np.array(flatten_img_list)
       new_flatten_img_list.shape
```

```
[158]: (4824, 32640)
```

```
[159]: all_image_data = []
       for x in range(0,len(holder)):
           temp = np.concatenate((new_flatten_img_list[x],holder[x]))
           all_image_data.append(temp)
```

```
[160]: df_all_image_data = pd.DataFrame(all_image_data)
       df_all_image_data = df_all_image_data.fillna(0.0)
```

1.3 Data Split

3. Split the data into training and testing. Be sure to explain how you performed this operation and why you think it is reasonable to split this particular dataset this way.

Due to the small size of the dataset, 4824 images, we will split the data into Training, Validation, and Testing, with set sizes as 50%, 25%, and 25%, respectively. This will allow us to train with most of the data while validating, and testing will have a large enough distribution to ensure generalization.

[161]:
```
X = df_all_image_data.to_numpy()
y = df_image_data["Emotion"].to_numpy()

label_encoder = LabelEncoder()
int_encoded = label_encoder.fit_transform(y)
oh_encoder = OneHotEncoder(sparse=False)
int_encoded = int_encoded.reshape(len(int_encoded), 1)
y_oh = oh_encoder.fit_transform(int_encoded)
```

[162]:
```
if len(df_image_data.columns) == 7:
    df_oh = pd.DataFrame(y_oh)
    df_image_data = pd.concat([df_image_data, df_oh], axis=1)


X_train, X_test, y_train, y_test = train_test_split(X, y_oh, test_size=0.25,␣
↪random_state=123321)
X_train, X_validate, y_train, y_validate = train_test_split(X_train, y_train,␣
↪test_size=0.333, random_state=123321, stratify=y_train)


print(f"X_train:{X_train.shape}")
print(f"y_train:{y_train.shape}")
print(f"X_validate:{X_validate.shape}")
print(f"y_validate:{y_validate.shape}")
print(f"X_test:{X_test.shape}")
print(f"y_test:{y_test.shape}")
```

```
X_train:(2413, 34044)
y_train:(2413, 8)
X_validate:(1205, 34044)
y_validate:(1205, 8)
X_test:(1206, 34044)
y_test:(1206, 8)
```

1.4 Train General Model

4. Train a general model (or per task model) to perform the classification tasks. That is, a general model uses all modalities and all tasks should combined into a single classification task (if possible). Alternatively, you could create a model for each specific task (if the general is not possible to build). For a task specific model, each task would be classified with its own feed-forward model.

1.4.1 MLP for Image Data

We choose to use a multilayer perceptron (MLP) as our architecture for the general model. We wanted to fuse the data as early as possible, which would be at the input layer of the model. We considered using a CNN architecture. However, a CNN works best to capture the spatial and temporal dependencies, and fusing the modalities will ruin the spatial dependencies of the image, potentially leading to a poor classification.

[22]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
generalized_mlp = Sequential()
generalized_mlp.add(Dense(input_dim=X_train.shape[1], units=2053,
                            kernel_initializer= tf.keras.initializers.
 ↪GlorotUniform(),
                            kernel_regularizer= tf.keras.regularizers.
 ↪l1_l2(l1=1e-5, l2=1e-4),
                            bias_regularizer= tf.keras.regularizers.l2(1e-4),
                            activity_regularizer= tf.keras.regularizers.
 ↪l2(1e-5),
                            activation='relu',
                            name='input_dense2053'))

generalized_mlp.add(Dense(units=991,
                            kernel_initializer=tf.keras.initializers.
 ↪GlorotUniform(),
                            activation='relu',
                            name='mid_dense991'))

generalized_mlp.add(Dense(units=381,
                            kernel_initializer=tf.keras.initializers.
 ↪GlorotUniform(),
                            activation='relu',
                            name='mid_dense381'))

generalized_mlp.add(Dense(units=109,
                            kernel_initializer=tf.keras.initializers.
 ↪GlorotUniform(),
                            activation='relu',
                            name='mid_dense109'))

generalized_mlp.add(Dense(units=31,
                            kernel_initializer=tf.keras.initializers.
 ↪GlorotUniform(),
                            activation='relu',
                            name='mid_dense31'))

generalized_mlp.add(Dense(units = len(EMOTIONAL_CATEGORIES),
```

```
                             activation='sigmoid',
                             name='output_layer'))

generalized_mlp.add(Activation('softmax'))


generalized_mlp.compile(loss='categorical_crossentropy',
                        optimizer= tf.keras.optimizers.RMSprop(learning_rate =␣
 ↪0.09973),
                        metrics=['accuracy', 'categorical_accuracy'])

history_mlp = generalized_mlp.fit(X_train,
                    y_train,
                    batch_size = 32,
                    epochs=20,
                    shuffle=True,
                    verbose=1,
                    validation_data=(X_validate,y_validate),
                    callbacks=[tf.keras.callbacks.
 ↪ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.
 ↪0001)])

y_hat_general = generalized_mlp.predict(X_test)
```

```
Epoch 1/20
76/76 [==============================] - 101s 1s/step - loss: 2804.5864 -
accuracy: 0.1289 - categorical_accuracy: 0.1289 - val_loss: 102.0862 -
val_accuracy: 0.1278 - val_categorical_accuracy: 0.1278 - lr: 0.0997
Epoch 2/20
76/76 [==============================] - 100s 1s/step - loss: 127.9403 -
accuracy: 0.1276 - categorical_accuracy: 0.1276 - val_loss: 152.4313 -
val_accuracy: 0.1278 - val_categorical_accuracy: 0.1278 - lr: 0.0997
Epoch 3/20
76/76 [==============================] - 99s 1s/step - loss: 144.8363 -
accuracy: 0.1276 - categorical_accuracy: 0.1276 - val_loss: 147.5046 -
val_accuracy: 0.1278 - val_categorical_accuracy: 0.1278 - lr: 0.0997
Epoch 4/20
76/76 [==============================] - 100s 1s/step - loss: 145.5036 -
accuracy: 0.1276 - categorical_accuracy: 0.1276 - val_loss: 150.3421 -
val_accuracy: 0.1278 - val_categorical_accuracy: 0.1278 - lr: 0.0997
Epoch 5/20
76/76 [==============================] - 100s 1s/step - loss: 146.1557 -
accuracy: 0.1276 - categorical_accuracy: 0.1276 - val_loss: 140.4621 -
val_accuracy: 0.1278 - val_categorical_accuracy: 0.1278 - lr: 0.0997
Epoch 6/20
76/76 [==============================] - 100s 1s/step - loss: 146.2717 -
accuracy: 0.1276 - categorical_accuracy: 0.1276 - val_loss: 146.8741 -
```

```
val_accuracy: 0.1278 - val_categorical_accuracy: 0.1278 - lr: 0.0997
Epoch 7/20
76/76 [==============================] - 100s 1s/step - loss: 35.3455 -
accuracy: 0.1276 - categorical_accuracy: 0.1276 - val_loss: 17.6022 -
val_accuracy: 0.1278 - val_categorical_accuracy: 0.1278 - lr: 0.0199
Epoch 8/20
76/76 [==============================] - 99s 1s/step - loss: 17.2850 - accuracy:
0.1276 - categorical_accuracy: 0.1276 - val_loss: 17.5083 - val_accuracy: 0.1278
- val_categorical_accuracy: 0.1278 - lr: 0.0199
Epoch 9/20
76/76 [==============================] - 100s 1s/step - loss: 17.4230 -
accuracy: 0.1276 - categorical_accuracy: 0.1276 - val_loss: 16.7502 -
val_accuracy: 0.1278 - val_categorical_accuracy: 0.1278 - lr: 0.0199
Epoch 10/20
76/76 [==============================] - 100s 1s/step - loss: 17.3257 -
accuracy: 0.1276 - categorical_accuracy: 0.1276 - val_loss: 17.6464 -
val_accuracy: 0.1278 - val_categorical_accuracy: 0.1278 - lr: 0.0199
Epoch 11/20
76/76 [==============================] - 100s 1s/step - loss: 17.2238 -
accuracy: 0.1276 - categorical_accuracy: 0.1276 - val_loss: 17.2529 -
val_accuracy: 0.1278 - val_categorical_accuracy: 0.1278 - lr: 0.0199
Epoch 12/20
76/76 [==============================] - 100s 1s/step - loss: 17.4053 -
accuracy: 0.1276 - categorical_accuracy: 0.1276 - val_loss: 16.7322 -
val_accuracy: 0.1278 - val_categorical_accuracy: 0.1278 - lr: 0.0199
Epoch 13/20
76/76 [==============================] - 100s 1s/step - loss: 17.2126 -
accuracy: 0.1276 - categorical_accuracy: 0.1276 - val_loss: 17.5129 -
val_accuracy: 0.1278 - val_categorical_accuracy: 0.1278 - lr: 0.0199
Epoch 14/20
76/76 [==============================] - 99s 1s/step - loss: 17.1965 - accuracy:
0.1276 - categorical_accuracy: 0.1276 - val_loss: 17.5472 - val_accuracy: 0.1278
- val_categorical_accuracy: 0.1278 - lr: 0.0199
Epoch 15/20
76/76 [==============================] - 99s 1s/step - loss: 17.2563 - accuracy:
0.1276 - categorical_accuracy: 0.1276 - val_loss: 17.3630 - val_accuracy: 0.1278
- val_categorical_accuracy: 0.1278 - lr: 0.0199
Epoch 16/20
76/76 [==============================] - 99s 1s/step - loss: 17.1991 - accuracy:
0.1276 - categorical_accuracy: 0.1276 - val_loss: 17.8918 - val_accuracy: 0.1278
- val_categorical_accuracy: 0.1278 - lr: 0.0199
Epoch 17/20
76/76 [==============================] - 99s 1s/step - loss: 17.3348 - accuracy:
0.1276 - categorical_accuracy: 0.1276 - val_loss: 17.2092 - val_accuracy: 0.1278
- val_categorical_accuracy: 0.1278 - lr: 0.0199
Epoch 18/20
76/76 [==============================] - 99s 1s/step - loss: 5.6157 - accuracy:
0.1276 - categorical_accuracy: 0.1276 - val_loss: 3.6991 - val_accuracy: 0.1278
```

```
- val_categorical_accuracy: 0.1278 - lr: 0.0040
Epoch 19/20
76/76 [==============================] - 99s 1s/step - loss: 3.6588 - accuracy:
0.1276 - categorical_accuracy: 0.1276 - val_loss: 3.6258 - val_accuracy: 0.1278
- val_categorical_accuracy: 0.1278 - lr: 0.0040
Epoch 20/20
76/76 [==============================] - 99s 1s/step - loss: 3.6252 - accuracy:
0.1276 - categorical_accuracy: 0.1276 - val_loss: 3.6142 - val_accuracy: 0.1278
- val_categorical_accuracy: 0.1278 - lr: 0.0040
```

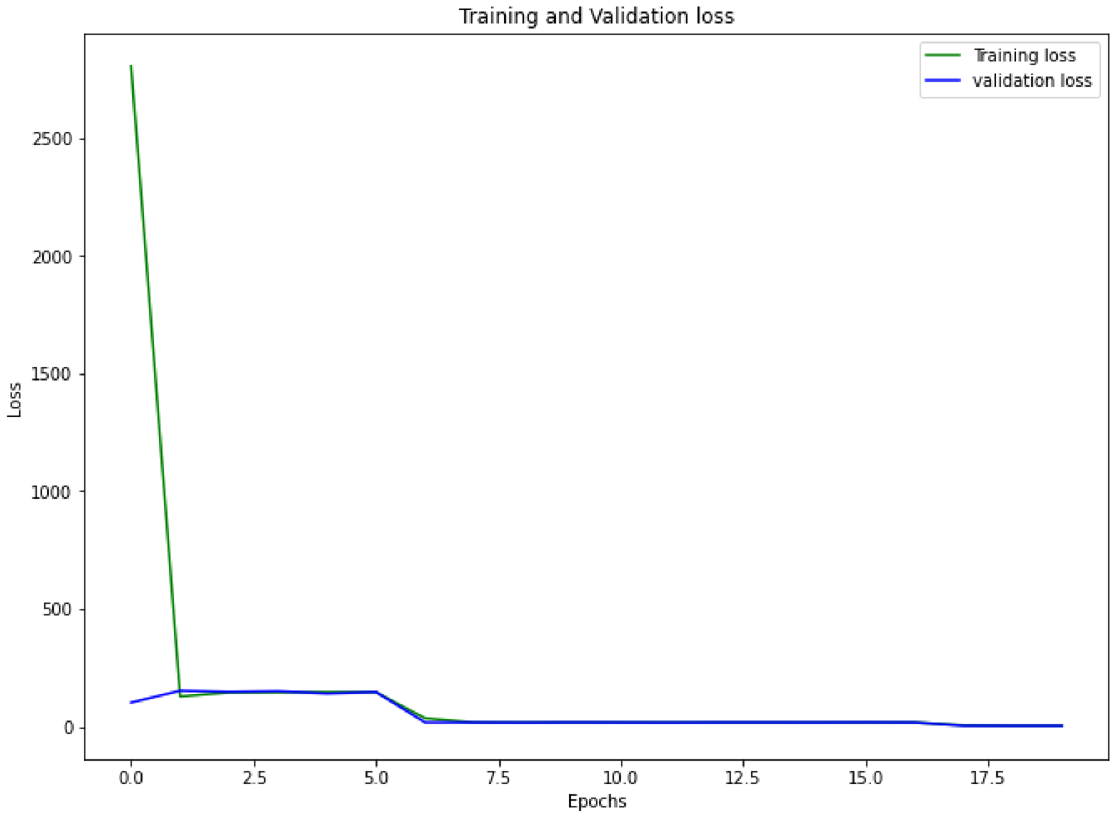
Using diffrent sets of iterations, we were only able to achieve an accuracy score of 0.1278, which is roughly a 1/8 chance of guessing the classification.

[24]:
```
def MakeDirectory(dir):
    dir_exits = os.path.exists(dir)
    if not dir_exits:
        os.mkdir(dir)
```

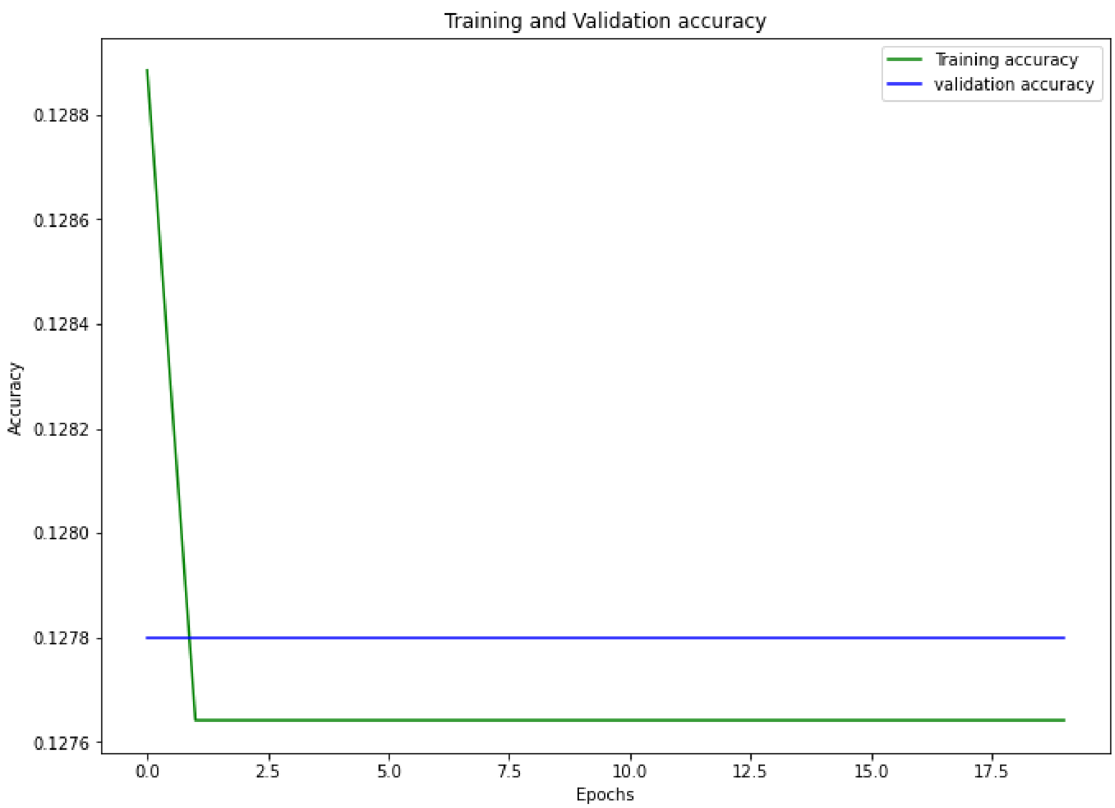
[25]:
```
generalized_mlp_dir = 'Data\\Models\\MLP1'
MakeDirectory(generalized_mlp_dir)
with open(os.path.join(generalized_mlp_dir, "hsitory.json"), 'wb') as file_pi:
        pickle.dump(history_mlp.history, file_pi)
generalized_mlp.save(os.path.join(generalized_mlp_dir, "MLP1.npy"))

#generalized_mlp_dir = 'Data\\Models\\MLP1'
#history_mlp = pickle.load(open(os.path.join(generalized_mlp_dir, "hsitory.
↪json")))
#generalized_mlp = load_model(os.path.join(generalized_mlp_dir, "MLP1.npy"))
```

```
INFO:tensorflow:Assets written to: Data\Models\MLP1\MLP1.npy\assets
```

[28]:
```
loss_train = history_mlp.history['loss']
loss_val = history_mlp.history['val_loss']
epochs = range(0,20)
plt.plot(epochs, loss_train, 'g', label='Training loss')
plt.plot(epochs, loss_val, 'b', label='validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



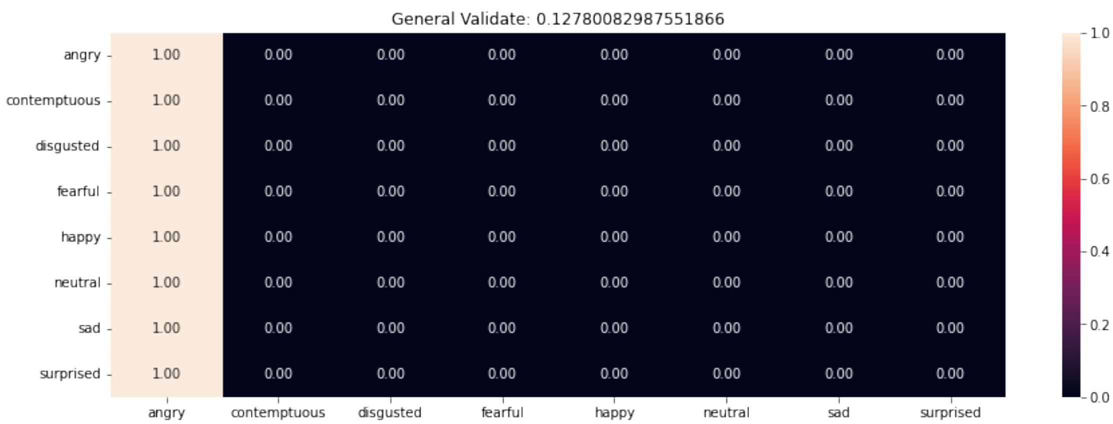
[29]:
```
loss_train = history_mlp.history['accuracy']
loss_val = history_mlp.history['val_accuracy']
epochs = range(0,20)
plt.plot(epochs, loss_train, 'g', label='Training accuracy')
plt.plot(epochs, loss_val, 'b', label='validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

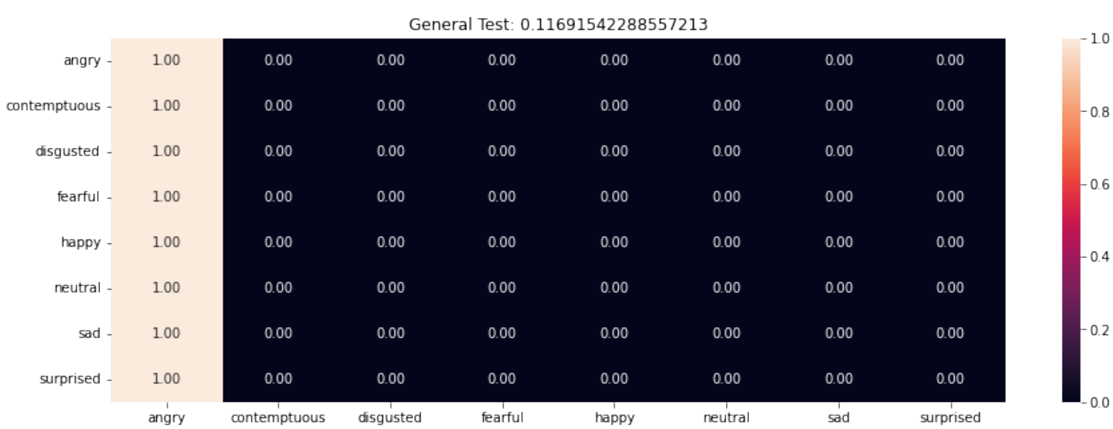


[74]:
```
confusion_matrix_cnn(generalized_mlp, X_train, y_train,␣
↪labels=EMOTIONAL_CATEGORIES, name="General Train: ")
confusion_matrix_cnn(generalized_mlp, X_validate, y_validate,␣
↪labels=EMOTIONAL_CATEGORIES, name="General Validate: ")
confusion_matrix_cnn(generalized_mlp, X_test, y_test,␣
↪labels=EMOTIONAL_CATEGORIES, name="General Test: ")
```

```
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
```

```
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
```





1.5 Train Multi Task-Model or Multi-Modal Model

5. Train a multi-task model (and/or) multi-modal model. You may use any method of multi-task or multi-modal training that you like.

[231]:
```
def confusion_matrix_cnn(cnn, X_test, y_test, labels='auto', name="CNN: "):
    plt.figure(figsize=(15,5))

    rounded_labels=np.argmax(y_test, axis=1)
    yhat_cnn = np.argmax(cnn.predict(X_test), axis=1)

    label_encoder = LabelEncoder()
    int_encoded = label_encoder.fit_transform(yhat_cnn)
    oh_encoder = OneHotEncoder(sparse=False)
    int_encoded = int_encoded.reshape(len(int_encoded), 1)
    yhat_oh_cnn = oh_encoder.fit_transform(int_encoded)

    acc_cnn = mt.accuracy_score(rounded_labels,yhat_cnn)
    roc_auc_cnn = mt.roc_auc_score(y_test,yhat_oh_cnn, multi_class='ovr')
    cm = mt.confusion_matrix(rounded_labels,yhat_cnn)
    cm = cm/np.sum(cm,axis=1)[:,np.newaxis]
    sns.heatmap(cm, annot=True, fmt='.2f',xticklabels=labels,yticklabels=labels)
    plt.title(f"{name} ACC:{acc_cnn}, ROCAUCP:{roc_auc_cnn}")
```

CNN for Image data

[232]:
```
#np_all_image_data = df_all_image_data.to_numpy()
#np_all_image_data[:,:new_flatten_img_list.shape[1]]

x_train_images = X_train[:,:new_flatten_img_list.shape[1]]
x_train_vectors = X_train[:,new_flatten_img_list.shape[1]+1:]

x_validate_images = X_validate[:,:new_flatten_img_list.shape[1]]
x_validate_vectors = X_validate[:,new_flatten_img_list.shape[1]+1:]

x_test_images = X_test[:,:new_flatten_img_list.shape[1]]
x_test_vectors = X_test[:,new_flatten_img_list.shape[1]+1:]

rxti = x_train_images.reshape((x_train_images.shape[0], MIN_X, MIN_Y, 3))
rxvi = x_validate_images.reshape((x_validate_images.shape[0], MIN_X, MIN_Y, 3))
rxei = x_test_images.reshape((x_test_images.shape[0], MIN_X, MIN_Y, 3))

print(f"x_train_images:{x_train_images.shape}")
print(f"x_train_vectors:{x_train_vectors.shape}")
print(f"x_validate_images:{x_validate_images.shape}")
print(f"x_validate_vectors:{x_validate_vectors.shape}")
print(f"x_test_images:{x_test_images.shape}")
print(f"x_test_vectors:{x_test_vectors.shape}")
```

```
x_train_images:(2413, 32640)
x_train_vectors:(2413, 1403)
x_validate_images:(1205, 32640)
x_validate_vectors:(1205, 1403)
x_test_images:(1206, 32640)
x_test_vectors:(1206, 1403)
```

[220]:
```
conv_model = models.Sequential()
conv_model.add(layers.Conv2D(16, (3, 3), activation="relu", input_shape=(MIN_X,␣
↪MIN_Y, 3)))
conv_model.add(layers.BatchNormalization(axis=-1))
conv_model.add(layers.MaxPooling2D(2, 2))
conv_model.add(layers.Conv2D(32, (3, 3), activation="relu"))
conv_model.add(layers.BatchNormalization(axis=-1))
conv_model.add(layers.MaxPooling2D(2, 2))
conv_model.add(layers.Conv2D(64, (3, 3), activation="relu"))
conv_model.add(layers.BatchNormalization(axis=-1))
conv_model.add(layers.MaxPooling2D(2, 2))
conv_model.add(layers.Flatten())
conv_model.add(layers.Dense(16, activation="relu"))
conv_model.add(layers.BatchNormalization(axis=-1))
conv_model.add(layers.Dropout(0.2))
conv_model.add(layers.Dense(8, activation="softmax"))

conv_model.summary()
conv_model.compile(optimizer="rmsprop", loss="categorical_crossentropy",␣
↪metrics=[
    tf.keras.metrics.CategoricalAccuracy(),
    tf.keras.metrics.AUC()
])


#history_image_cnn = conv_model.fit(
#    x= rxti,
#    y=y_train,
#    batch_size = 32,
#    steps_per_epoch = 50,
#    epochs = 50,
#    validation_data = (rxvi, y_validate),
#    verbose = True)
#
#y_hat_image_cnn = conv_model.predict(rxei)
```

```
Model: "sequential_39"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 83, 126, 16)       448
```

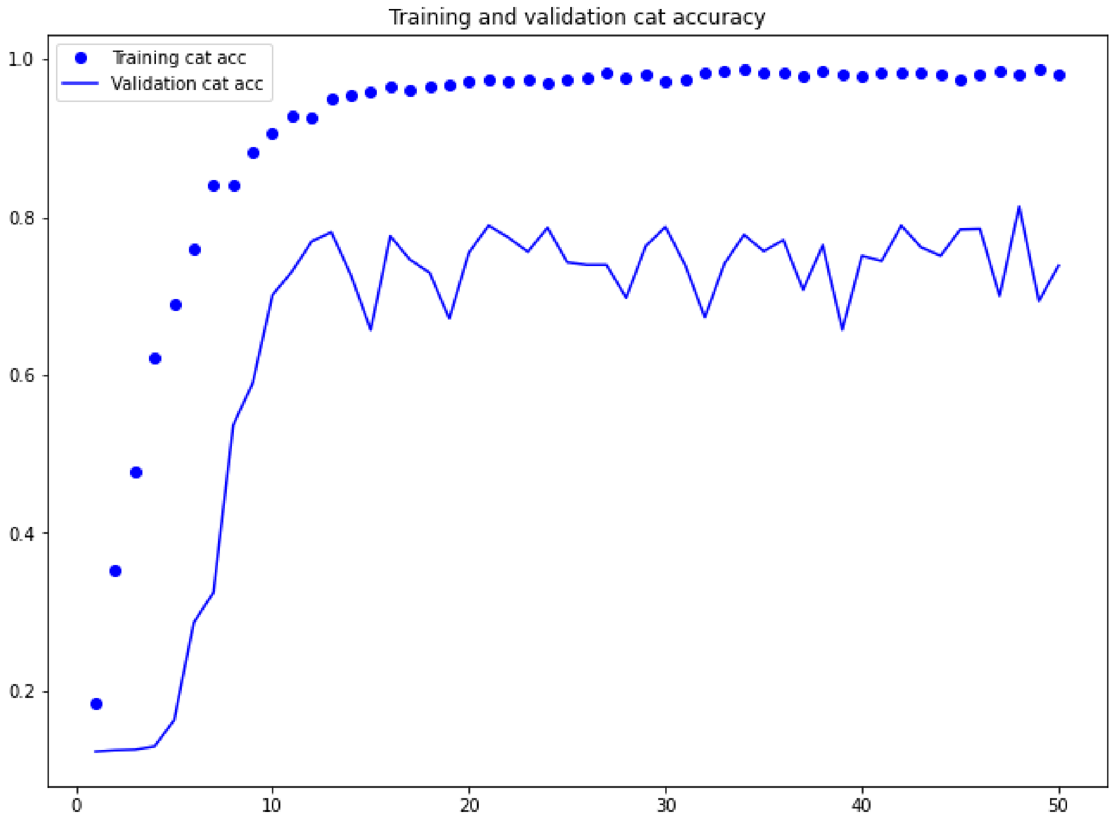
```
 batch_normalization_24 (Bat  (None, 83, 126, 16)      64
 chNormalization)

 max_pooling2d_18 (MaxPoolin  (None, 41, 63, 16)       0
 g2D)

 conv2d_19 (Conv2D)          (None, 39, 61, 32)        4640

 batch_normalization_25 (Bat  (None, 39, 61, 32)       128
 chNormalization)

 max_pooling2d_19 (MaxPoolin  (None, 19, 30, 32)       0
 g2D)

 conv2d_20 (Conv2D)          (None, 17, 28, 64)        18496

 batch_normalization_26 (Bat  (None, 17, 28, 64)       256
 chNormalization)

 max_pooling2d_20 (MaxPoolin  (None, 8, 14, 64)        0
 g2D)

 flatten_6 (Flatten)         (None, 7168)              0

 dense_30 (Dense)            (None, 16)                114704

 batch_normalization_27 (Bat  (None, 16)               64
 chNormalization)

 dropout_6 (Dropout)         (None, 16)                0

 dense_31 (Dense)            (None, 8)                 136

=================================================================
Total params: 138,936
Trainable params: 138,680
Non-trainable params: 256
_________________________________________________________________
```

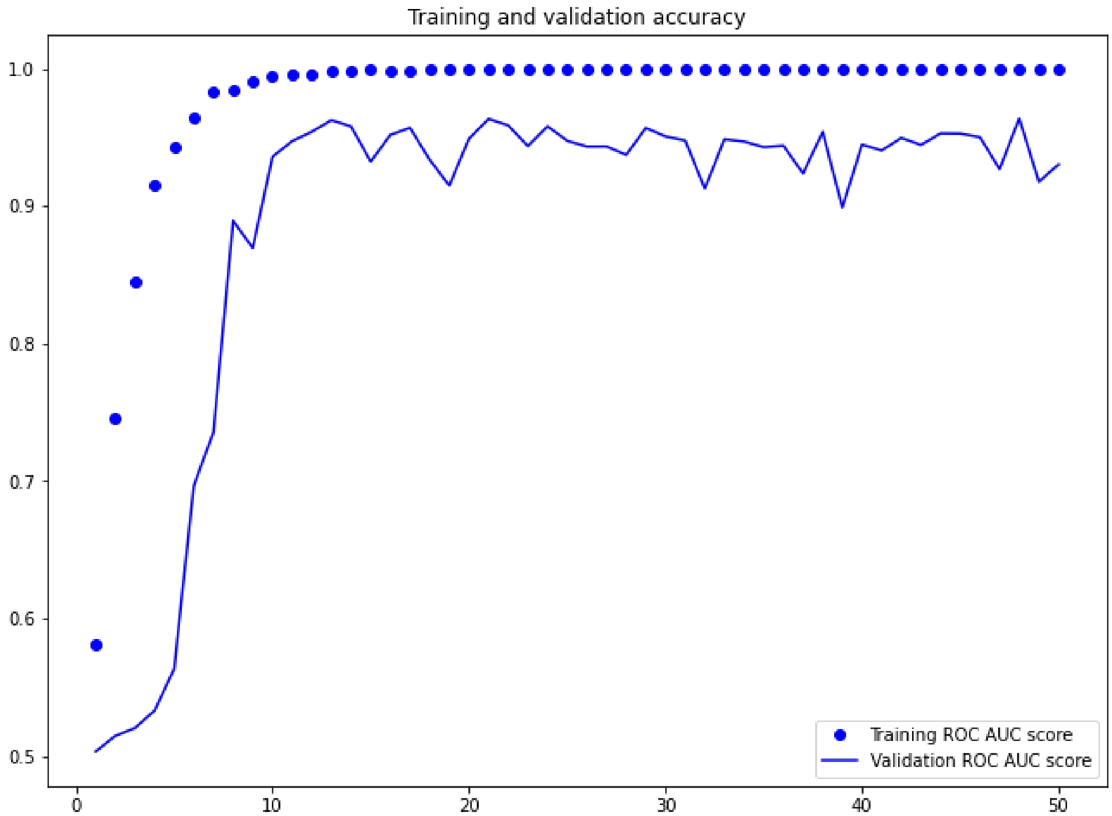
[201]:

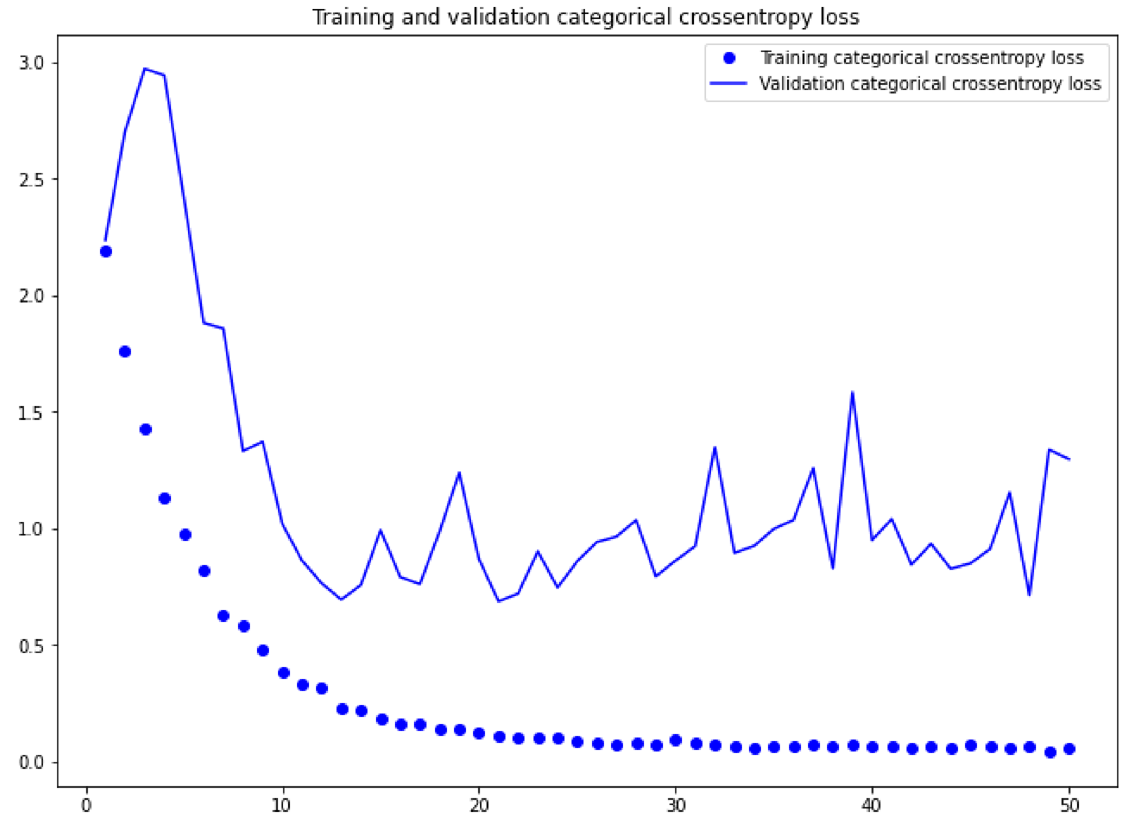
```
model1_dir = 'Data\\Models\\CN1'
MakeDirectory(model1_dir)
with open(os.path.join(model1_dir, "hsitory.json"), 'wb') as file_pi:
        pickle.dump(history_image_cnn.history, file_pi)
conv_model.save(os.path.join(model1_dir, "CN1.npy"))

#model1_dir = 'Data\\Models\\CN1'
```

```
#history = pickle.load(open(os.path.join(model1_dir, "hsitory.json")), "rb")
#conv_model = load_model(os.path.join(model1_dir, "CN1.npy"))

#Code based on "F. Chollet, Deep learning with Python. 2021" work in Listing 5.
↪10 on page 137
acc = history_image_cnn.history['auc_8']
val_acc = history_image_cnn.history['val_auc_8']
cat_acc = history_image_cnn.history['categorical_accuracy']
val_cat_acc = history_image_cnn.history['val_categorical_accuracy']
loss = history_image_cnn.history['loss']
val_loss = history_image_cnn.history['val_loss']

epochs = range(1, len(acc) + 1)


plt.plot(epochs, cat_acc, 'bo', label='Training cat acc')
plt.plot(epochs, val_cat_acc, 'b', label='Validation cat acc')
plt.title('Training and validation cat accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, acc, 'bo', label='Training ROC AUC score')
plt.plot(epochs, val_acc, 'b', label='Validation ROC AUC score')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training categorical crossentropy loss')
plt.plot(epochs, val_loss, 'b', label='Validation categorical crossentropy␣
↪loss')
plt.title('Training and validation categorical crossentropy loss')
plt.legend()

plt.figure()

plt.show()

confusion_matrix_cnn(conv_model, rxti, y_train, labels=EMOTIONAL_CATEGORIES,␣
↪name= "CNN1 Train: ")
confusion_matrix_cnn(conv_model, rxvi, y_validate, labels=EMOTIONAL_CATEGORIES,␣
↪name= "CNN1 Val:")
confusion_matrix_cnn(conv_model, rxei, y_test, labels=EMOTIONAL_CATEGORIES,␣
↪name= "CNN1 Test:")
```

```
INFO:tensorflow:Assets written to: Data\Models\CN1\CN1.npy\assets
```

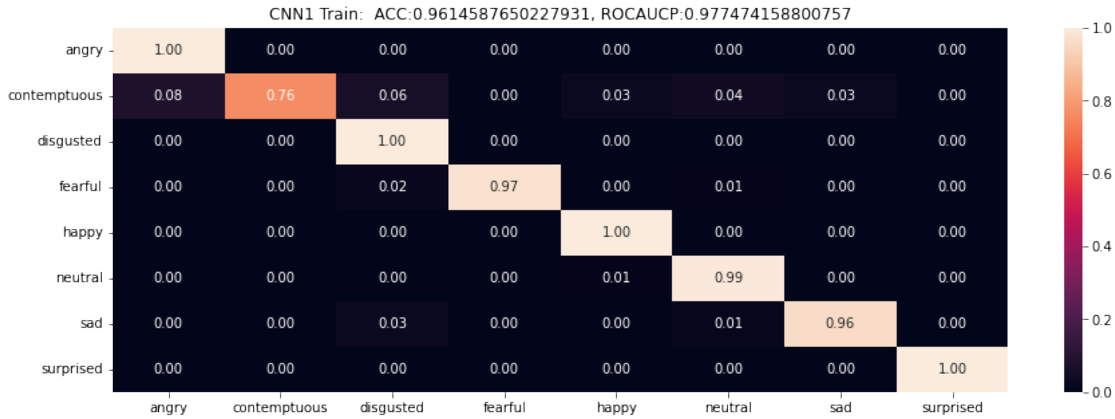

Training and validation cat accuracy
Training cat acc
Validation cat acc
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50

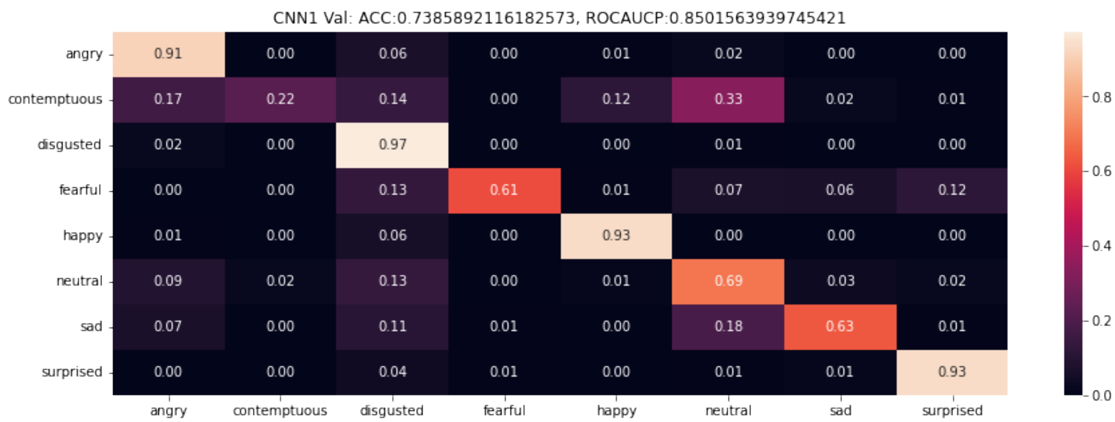

Training and validation accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0
10
20
30
40
50
Training ROC AUC score
Validation ROC AUC score

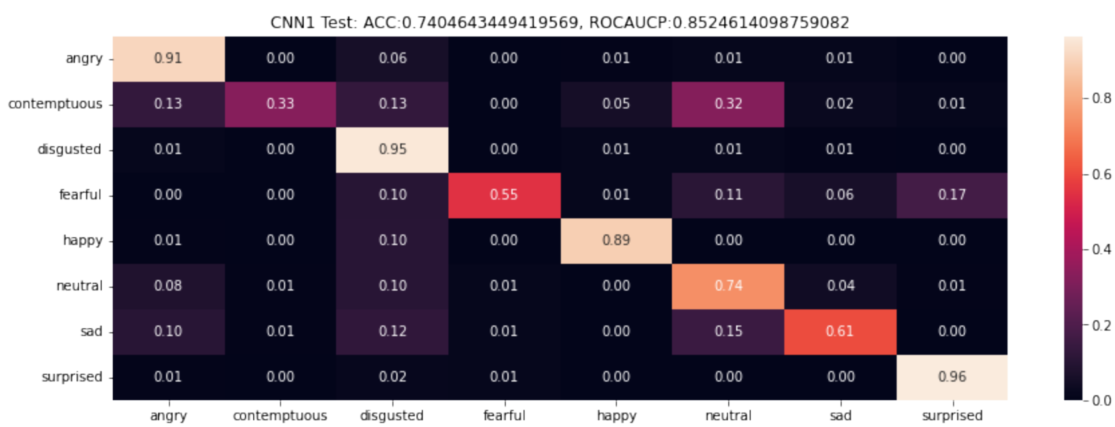



```
<Figure size 792x576 with 0 Axes>
```

```
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
```







MLP for vector data

[221]:
```
vector_mlp = Sequential()
vector_mlp.add(Dense(input_dim=x_train_vectors.shape[1], units=991,
                             kernel_initializer= tf.keras.initializers.
 ↪GlorotUniform(),
                             kernel_regularizer= tf.keras.regularizers.
 ↪l1_l2(l1=1e-5, l2=1e-4),
                             bias_regularizer= tf.keras.regularizers.l2(1e-4),
                             activity_regularizer= tf.keras.regularizers.
 ↪l2(1e-5),
                             activation='relu',
                             name='input_dense2053'))

vector_mlp.add(Dense(units=381,
                             kernel_initializer=tf.keras.initializers.
 ↪GlorotUniform(),
                             kernel_regularizer= tf.keras.regularizers.
 ↪l1_l2(l1=1e-5, l2=1e-4),
                             bias_regularizer= tf.keras.regularizers.l2(1e-4),
                             activity_regularizer= tf.keras.regularizers.
 ↪l2(1e-5),
                             activation='relu',
                             name='mid_dense991'))

vector_mlp.add(Dense(units=109,
                             kernel_initializer=tf.keras.initializers.
 ↪GlorotUniform(),
                             kernel_regularizer= tf.keras.regularizers.
 ↪l1_l2(l1=1e-5, l2=1e-4),
                             bias_regularizer= tf.keras.regularizers.l2(1e-4),
                             activity_regularizer= tf.keras.regularizers.
 ↪l2(1e-5),
                             activation='relu',
                             name='mid_dense381'))

vector_mlp.add(Dense(units=31,
                             kernel_initializer=tf.keras.initializers.
 ↪GlorotUniform(),
                             kernel_regularizer= tf.keras.regularizers.
 ↪l1_l2(l1=1e-5, l2=1e-4),
                             bias_regularizer= tf.keras.regularizers.l2(1e-4),
                             activity_regularizer= tf.keras.regularizers.
 ↪l2(1e-5),
                             activation='relu',
                             name='mid_dense109'))
```

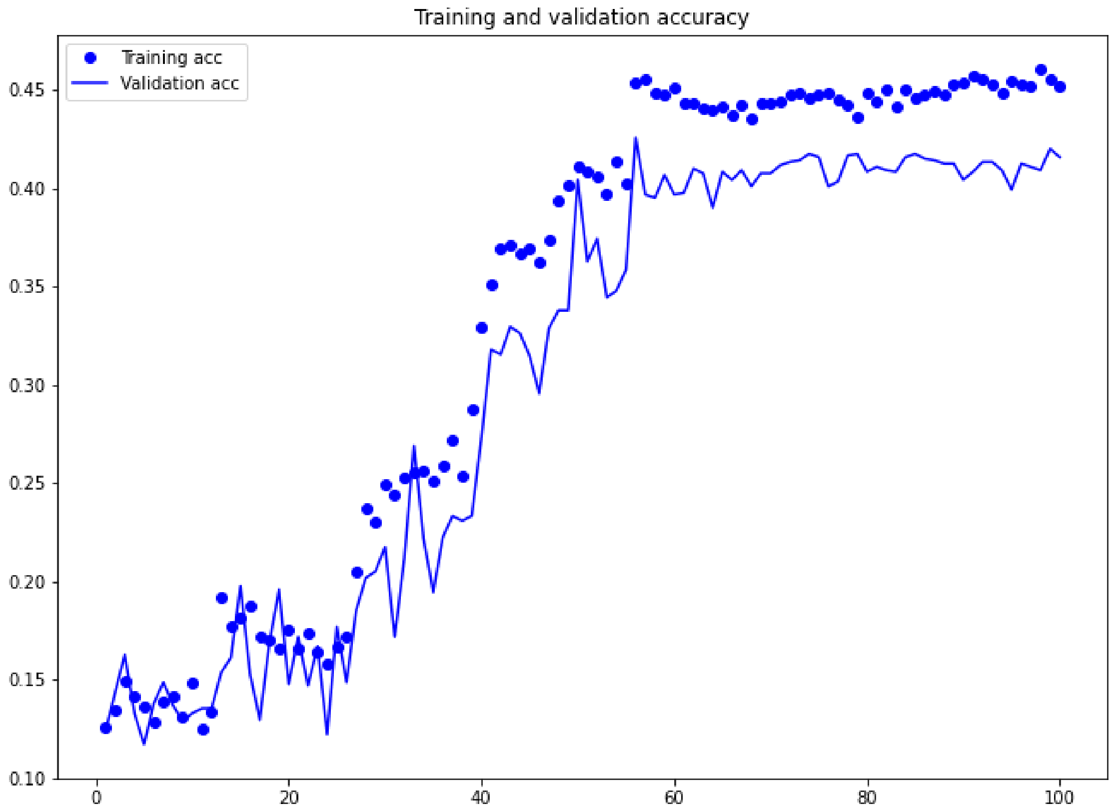
```
vector_mlp.add(Dense(units = len(EMOTIONAL_CATEGORIES),
                            kernel_regularizer= tf.keras.regularizers.
 ↪l1_l2(l1=1e-5, l2=1e-4),
                            bias_regularizer= tf.keras.regularizers.l2(1e-4),
                            activity_regularizer= tf.keras.regularizers.
 ↪l2(1e-5),
                            activation='sigmoid',
                            name='output_layer'))

vector_mlp.add(Activation('softmax'))


vector_mlp.compile(loss='categorical_crossentropy',
                        optimizer= tf.keras.optimizers.RMSprop(learning_rate =␣
 ↪0.09973),
                        metrics=['accuracy', 'categorical_accuracy'])

#history_vector_mlp = vector_mlp.fit(x_train_vectors,
#                         y_train,
#                         batch_size = 32,
#                         epochs=100,
#                         shuffle=True,
#                         verbose=1,
#                         validation_data=(x_validate_vectors,y_validate),
#                         callbacks=[tf.keras.callbacks.
 ↪ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.
 ↪0001)])

#y_hat_vector_mlp = vector_mlp.predict(x_train_vectors)
```

[202]:

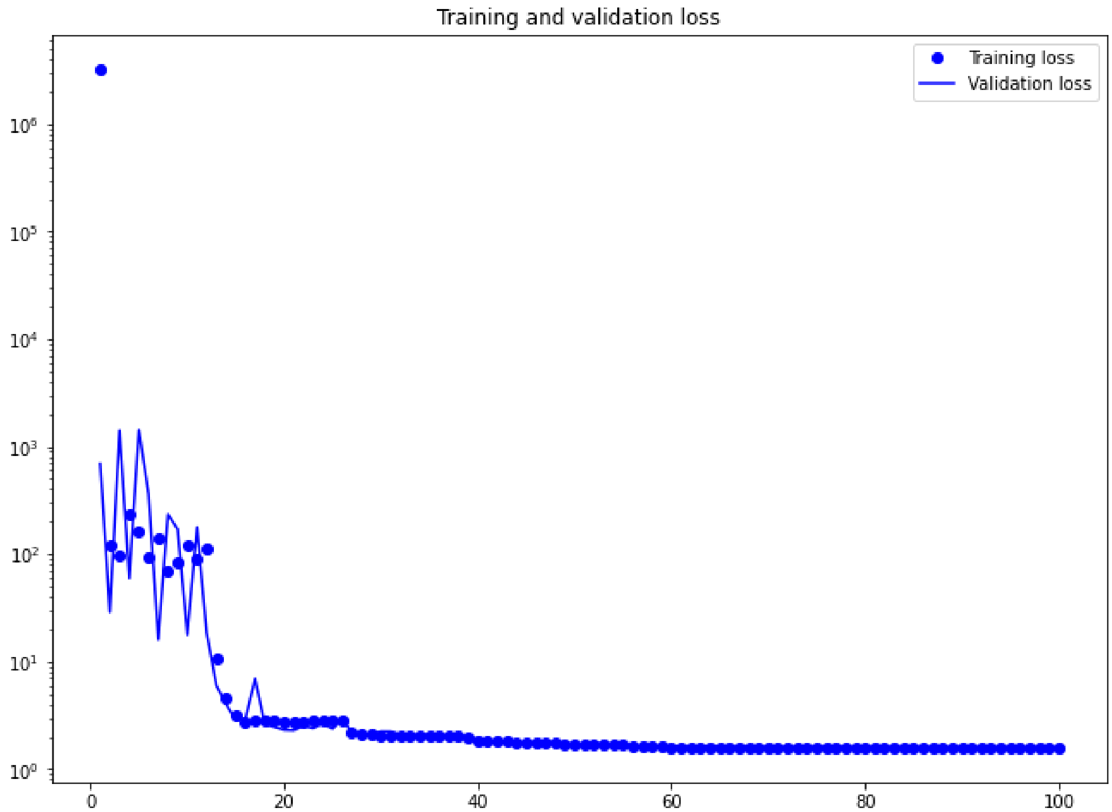
```
model2_dir = 'Data\\Models\\MLP2'
MakeDirectory(model2_dir)
with open(os.path.join(model2_dir, "hsitory.json"), 'wb') as file_pi:
        pickle.dump(history_vector_mlp.history, file_pi)
vector_mlp.save(os.path.join(model2_dir, "MLP2.npy"))

#model2_dir = 'Data\\Models\\CN1'
#history = pickle.load(open(os.path.join(model2_dir, "hsitory.json")), "rb")
#vector_mlp = load_model(os.path.join(model2_dir, "CN1.npy"))

#Code based on "F. Chollet, Deep learning with Python. 2021" work in Listing 5.
 ↪10 on page 137
acc = history_vector_mlp.history['accuracy']
val_acc = history_vector_mlp.history['val_accuracy']
cat_acc = history_vector_mlp.history['categorical_accuracy']
val_cat_acc = history_vector_mlp.history['val_categorical_accuracy']
loss = history_vector_mlp.history['loss']
```

```
val_loss = history_vector_mlp.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, cat_acc, 'bo', label='Training cat acc')
plt.plot(epochs, val_cat_acc, 'b', label='Validation cat acc')
plt.title('Training and validation cat accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.yscale("log")
plt.legend()

plt.figure()

plt.show()


confusion_matrix_cnn(vector_mlp, x_train_vectors, y_train,␣
 ↪labels=EMOTIONAL_CATEGORIES, name='MLP2 train: ')
confusion_matrix_cnn(vector_mlp, x_validate_vectors, y_validate,␣
 ↪labels=EMOTIONAL_CATEGORIES, name='MLP2 val: ')
confusion_matrix_cnn(vector_mlp, x_test_vectors, y_test,␣
 ↪labels=EMOTIONAL_CATEGORIES, name='MLP2 test: ')
```

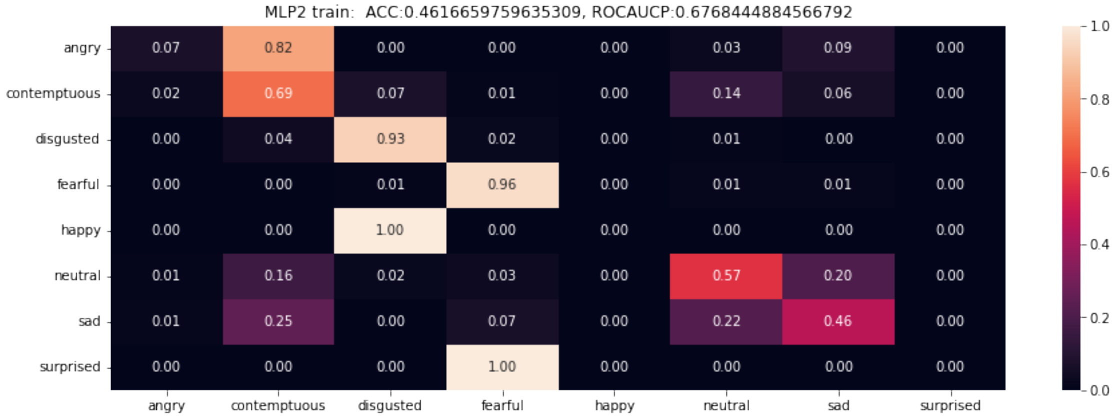
```
INFO:tensorflow:Assets written to: Data\Models\MLP2\MLP2.npy\assets
```

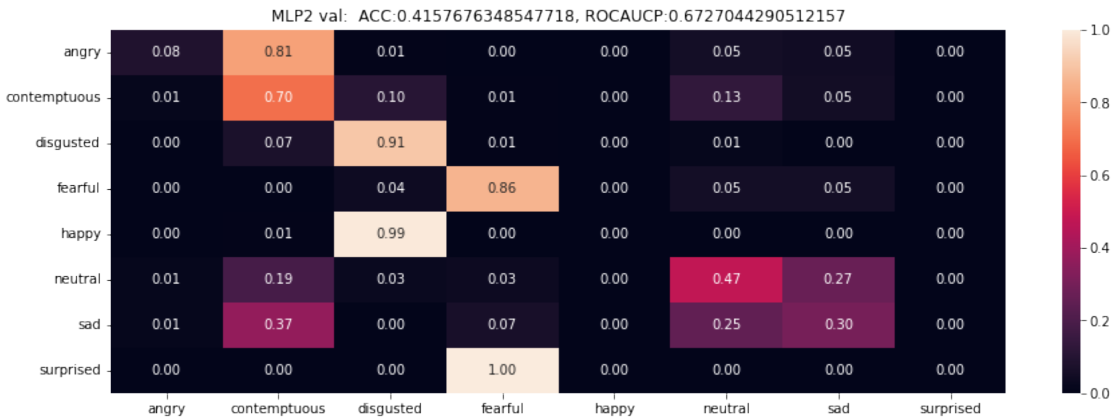

Training and validation accuracy
Training acc
Validation acc
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
20
40
60
80
100

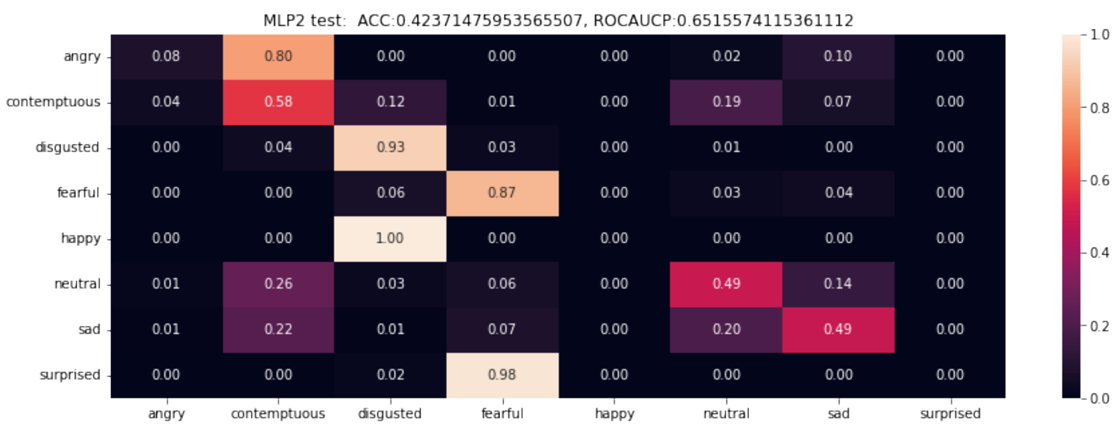

Training and validation cat accuracy
Training cat acc
Validation cat acc
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
20
40
60
80
100




```
<Figure size 792x576 with 0 Axes>
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
```







Mix CNN and MLP

[34]:
```
tf.config.experimental_run_functions_eagerly(True)
tf.config.run_functions_eagerly(True)
```

```
WARNING:tensorflow:From
C:\Users\JAKEKL~1\AppData\Local\Temp/ipykernel_65644/403206910.py:1:
experimental_run_functions_eagerly (from tensorflow.python.eager.def_function)
is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.config.run_functions_eagerly` instead of the experimental version.
```

[54]:
```
# Combining the two together
combined_input = tf.keras.layers.concatenate([conv_model.output,vector_mlp.
↪output])
mm1_out = Dense(16, activation="relu")(combined_input)
mm1_out = Dense(8, activation="softmax")(mm1_out)

multimodal_model1 = tf.keras.models.Model(inputs= [conv_model.input,vector_mlp.
↪input], outputs= mm1_out)

multimodal_model1.compile(optimizer="rmsprop", loss="categorical_crossentropy",␣
↪metrics=[
    tf.keras.metrics.CategoricalAccuracy(),
    tf.keras.metrics.AUC()
])
```

[55]:
```
history_mm1 = multimodal_model1.fit(x=[rxti, x_train_vectors], y=y_train,
                                    validation_data=([rxvi,␣
↪x_validate_vectors], y_validate),
                                    epochs=50,
                                    batch_size=32)
```

```
Epoch 1/50

C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(

76/76 [==============================] - 25s 324ms/step - loss: 2.2664 -
categorical_accuracy: 0.1757 - auc_4: 0.5372 - val_loss: 2.1072 -
val_categorical_accuracy: 0.1212 - val_auc_4: 0.5252
Epoch 2/50
76/76 [==============================] - 24s 320ms/step - loss: 2.0406 -
categorical_accuracy: 0.2246 - auc_4: 0.6473 - val_loss: 2.1208 -
val_categorical_accuracy: 0.1212 - val_auc_4: 0.5140
Epoch 3/50
```

```
76/76 [==============================] - 24s 321ms/step - loss: 1.9615 -
categorical_accuracy: 0.2561 - auc_4: 0.7276 - val_loss: 2.0847 -
val_categorical_accuracy: 0.1436 - val_auc_4: 0.5461
Epoch 4/50
76/76 [==============================] - 25s 325ms/step - loss: 1.8615 -
categorical_accuracy: 0.3058 - auc_4: 0.7828 - val_loss: 1.9347 -
val_categorical_accuracy: 0.2448 - val_auc_4: 0.6875
Epoch 5/50
76/76 [==============================] - 25s 333ms/step - loss: 1.7780 -
categorical_accuracy: 0.3469 - auc_4: 0.8069 - val_loss: 1.8614 -
val_categorical_accuracy: 0.2581 - val_auc_4: 0.7203
Epoch 6/50
76/76 [==============================] - 25s 326ms/step - loss: 1.6987 -
categorical_accuracy: 0.3692 - auc_4: 0.8262 - val_loss: 1.8000 -
val_categorical_accuracy: 0.2664 - val_auc_4: 0.7501
Epoch 7/50
76/76 [==============================] - 25s 329ms/step - loss: 1.6125 -
categorical_accuracy: 0.3796 - auc_4: 0.8472 - val_loss: 1.6663 -
val_categorical_accuracy: 0.3527 - val_auc_4: 0.8217
Epoch 8/50
76/76 [==============================] - 24s 310ms/step - loss: 1.5486 -
categorical_accuracy: 0.3941 - auc_4: 0.8551 - val_loss: 1.6972 -
val_categorical_accuracy: 0.3436 - val_auc_4: 0.7840
Epoch 9/50
76/76 [==============================] - 23s 299ms/step - loss: 1.4810 -
categorical_accuracy: 0.4049 - auc_4: 0.8677 - val_loss: 1.5323 -
val_categorical_accuracy: 0.4266 - val_auc_4: 0.8517
Epoch 10/50
76/76 [==============================] - 23s 300ms/step - loss: 1.4211 -
categorical_accuracy: 0.4418 - auc_4: 0.8799 - val_loss: 1.7886 -
val_categorical_accuracy: 0.2863 - val_auc_4: 0.7486
Epoch 11/50
76/76 [==============================] - 23s 308ms/step - loss: 1.3481 -
categorical_accuracy: 0.4741 - auc_4: 0.8986 - val_loss: 1.4720 -
val_categorical_accuracy: 0.4672 - val_auc_4: 0.8626
Epoch 12/50
76/76 [==============================] - 24s 310ms/step - loss: 1.2774 -
categorical_accuracy: 0.5139 - auc_4: 0.9127 - val_loss: 1.4462 -
val_categorical_accuracy: 0.4207 - val_auc_4: 0.8614
Epoch 13/50
76/76 [==============================] - 23s 306ms/step - loss: 1.1910 -
categorical_accuracy: 0.5769 - auc_4: 0.9281 - val_loss: 1.2491 -
val_categorical_accuracy: 0.5427 - val_auc_4: 0.9088
Epoch 14/50
76/76 [==============================] - 24s 312ms/step - loss: 1.1090 -
categorical_accuracy: 0.6022 - auc_4: 0.9374 - val_loss: 1.2375 -
val_categorical_accuracy: 0.5610 - val_auc_4: 0.9077
Epoch 15/50
```

```
76/76 [==============================] - 24s 310ms/step - loss: 1.0073 -
categorical_accuracy: 0.6324 - auc_4: 0.9468 - val_loss: 1.4135 -
val_categorical_accuracy: 0.4324 - val_auc_4: 0.8603
Epoch 16/50
76/76 [==============================] - 23s 309ms/step - loss: 0.9338 -
categorical_accuracy: 0.6328 - auc_4: 0.9515 - val_loss: 1.1681 -
val_categorical_accuracy: 0.5104 - val_auc_4: 0.9111
Epoch 17/50
76/76 [==============================] - 24s 314ms/step - loss: 0.8829 -
categorical_accuracy: 0.6287 - auc_4: 0.9527 - val_loss: 1.0087 -
val_categorical_accuracy: 0.5668 - val_auc_4: 0.9311
Epoch 18/50
76/76 [==============================] - 24s 313ms/step - loss: 0.8075 -
categorical_accuracy: 0.6535 - auc_4: 0.9569 - val_loss: 1.1818 -
val_categorical_accuracy: 0.5220 - val_auc_4: 0.9026
Epoch 19/50
76/76 [==============================] - 23s 300ms/step - loss: 0.7550 -
categorical_accuracy: 0.6598 - auc_4: 0.9600 - val_loss: 1.3032 -
val_categorical_accuracy: 0.4896 - val_auc_4: 0.8874
Epoch 20/50
76/76 [==============================] - 23s 299ms/step - loss: 0.7308 -
categorical_accuracy: 0.6622 - auc_4: 0.9609 - val_loss: 1.2268 -
val_categorical_accuracy: 0.4996 - val_auc_4: 0.8995
Epoch 21/50
76/76 [==============================] - 23s 299ms/step - loss: 0.7106 -
categorical_accuracy: 0.6660 - auc_4: 0.9603 - val_loss: 1.1510 -
val_categorical_accuracy: 0.5054 - val_auc_4: 0.9050
Epoch 22/50
76/76 [==============================] - 23s 299ms/step - loss: 0.6708 -
categorical_accuracy: 0.6722 - auc_4: 0.9625 - val_loss: 1.1099 -
val_categorical_accuracy: 0.5510 - val_auc_4: 0.9166
Epoch 23/50
76/76 [==============================] - 23s 299ms/step - loss: 0.6561 -
categorical_accuracy: 0.6855 - auc_4: 0.9644 - val_loss: 0.8547 -
val_categorical_accuracy: 0.6382 - val_auc_4: 0.9476
Epoch 24/50
76/76 [==============================] - 23s 298ms/step - loss: 0.6524 -
categorical_accuracy: 0.6705 - auc_4: 0.9637 - val_loss: 1.2171 -
val_categorical_accuracy: 0.5278 - val_auc_4: 0.9067
Epoch 25/50
76/76 [==============================] - 23s 299ms/step - loss: 0.6427 -
categorical_accuracy: 0.6904 - auc_4: 0.9660 - val_loss: 1.3915 -
val_categorical_accuracy: 0.5071 - val_auc_4: 0.8842
Epoch 26/50
76/76 [==============================] - 24s 310ms/step - loss: 0.6239 -
categorical_accuracy: 0.6933 - auc_4: 0.9682 - val_loss: 1.1501 -
val_categorical_accuracy: 0.5892 - val_auc_4: 0.9190
Epoch 27/50
```

```
76/76 [==============================] - 24s 310ms/step - loss: 0.6198 -
categorical_accuracy: 0.6908 - auc_4: 0.9675 - val_loss: 0.9278 -
val_categorical_accuracy: 0.6174 - val_auc_4: 0.9406
Epoch 28/50
76/76 [==============================] - 24s 312ms/step - loss: 0.6090 -
categorical_accuracy: 0.6991 - auc_4: 0.9688 - val_loss: 1.6565 -
val_categorical_accuracy: 0.4954 - val_auc_4: 0.8474
Epoch 29/50
76/76 [==============================] - 24s 311ms/step - loss: 0.6190 -
categorical_accuracy: 0.6855 - auc_4: 0.9670 - val_loss: 0.9131 -
val_categorical_accuracy: 0.6091 - val_auc_4: 0.9418
Epoch 30/50
76/76 [==============================] - 24s 313ms/step - loss: 0.6127 -
categorical_accuracy: 0.7053 - auc_4: 0.9704 - val_loss: 0.9095 -
val_categorical_accuracy: 0.6133 - val_auc_4: 0.9446
Epoch 31/50
76/76 [==============================] - 23s 301ms/step - loss: 0.6235 -
categorical_accuracy: 0.7024 - auc_4: 0.9688 - val_loss: 0.8980 -
val_categorical_accuracy: 0.6324 - val_auc_4: 0.9443
Epoch 32/50
76/76 [==============================] - 23s 299ms/step - loss: 0.6131 -
categorical_accuracy: 0.6975 - auc_4: 0.9694 - val_loss: 0.8803 -
val_categorical_accuracy: 0.6656 - val_auc_4: 0.9470
Epoch 33/50
76/76 [==============================] - 23s 299ms/step - loss: 0.5669 -
categorical_accuracy: 0.7302 - auc_4: 0.9751 - val_loss: 0.9790 -
val_categorical_accuracy: 0.5726 - val_auc_4: 0.9328
Epoch 34/50
76/76 [==============================] - 23s 301ms/step - loss: 0.5916 -
categorical_accuracy: 0.7145 - auc_4: 0.9714 - val_loss: 1.3981 -
val_categorical_accuracy: 0.5253 - val_auc_4: 0.8896
Epoch 35/50
76/76 [==============================] - 23s 300ms/step - loss: 0.5777 -
categorical_accuracy: 0.7385 - auc_4: 0.9752 - val_loss: 1.3198 -
val_categorical_accuracy: 0.5361 - val_auc_4: 0.9036
Epoch 36/50
76/76 [==============================] - 23s 300ms/step - loss: 0.5713 -
categorical_accuracy: 0.7344 - auc_4: 0.9748 - val_loss: 1.0095 -
val_categorical_accuracy: 0.6656 - val_auc_4: 0.9376
Epoch 37/50
76/76 [==============================] - 23s 301ms/step - loss: 0.5794 -
categorical_accuracy: 0.7435 - auc_4: 0.9744 - val_loss: 0.8923 -
val_categorical_accuracy: 0.6481 - val_auc_4: 0.9441
Epoch 38/50
76/76 [==============================] - 23s 299ms/step - loss: 0.5606 -
categorical_accuracy: 0.7422 - auc_4: 0.9763 - val_loss: 1.3668 -
val_categorical_accuracy: 0.5278 - val_auc_4: 0.8932
Epoch 39/50
```

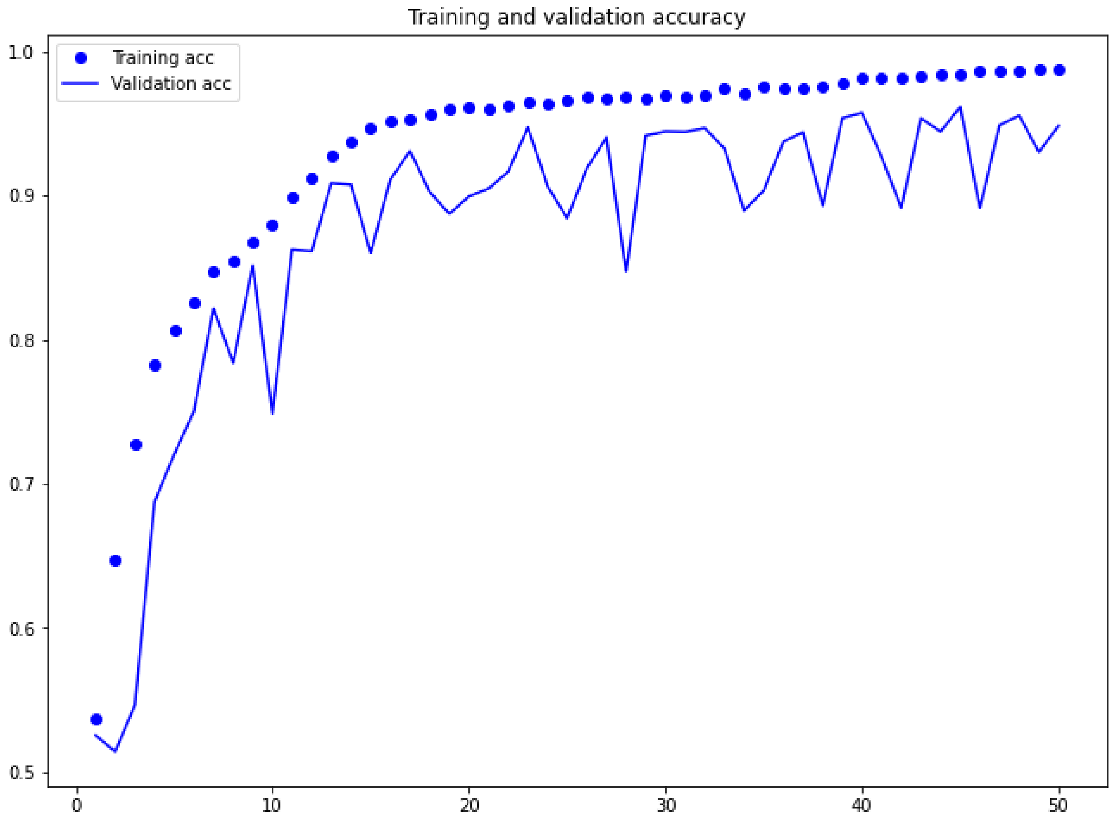
```
76/76 [==============================] - 25s 323ms/step - loss: 0.5438 -
categorical_accuracy: 0.7542 - auc_4: 0.9780 - val_loss: 0.8498 -
val_categorical_accuracy: 0.6954 - val_auc_4: 0.9539
Epoch 40/50
76/76 [==============================] - 24s 322ms/step - loss: 0.5244 -
categorical_accuracy: 0.7816 - auc_4: 0.9814 - val_loss: 0.7793 -
val_categorical_accuracy: 0.7145 - val_auc_4: 0.9576
Epoch 41/50
76/76 [==============================] - 24s 322ms/step - loss: 0.5137 -
categorical_accuracy: 0.7812 - auc_4: 0.9816 - val_loss: 1.0437 -
val_categorical_accuracy: 0.6224 - val_auc_4: 0.9268
Epoch 42/50
76/76 [==============================] - 23s 303ms/step - loss: 0.5058 -
categorical_accuracy: 0.7924 - auc_4: 0.9822 - val_loss: 1.4308 -
val_categorical_accuracy: 0.5851 - val_auc_4: 0.8914
Epoch 43/50
76/76 [==============================] - 23s 303ms/step - loss: 0.5105 -
categorical_accuracy: 0.8015 - auc_4: 0.9828 - val_loss: 0.8010 -
val_categorical_accuracy: 0.7004 - val_auc_4: 0.9538
Epoch 44/50
76/76 [==============================] - 23s 303ms/step - loss: 0.4885 -
categorical_accuracy: 0.8011 - auc_4: 0.9841 - val_loss: 0.9046 -
val_categorical_accuracy: 0.6697 - val_auc_4: 0.9445
Epoch 45/50
76/76 [==============================] - 23s 305ms/step - loss: 0.4742 -
categorical_accuracy: 0.8007 - auc_4: 0.9846 - val_loss: 0.7542 -
val_categorical_accuracy: 0.7402 - val_auc_4: 0.9617
Epoch 46/50
76/76 [==============================] - 23s 307ms/step - loss: 0.4589 -
categorical_accuracy: 0.8189 - auc_4: 0.9863 - val_loss: 1.4264 -
val_categorical_accuracy: 0.5361 - val_auc_4: 0.8916
Epoch 47/50
76/76 [==============================] - 23s 305ms/step - loss: 0.4581 -
categorical_accuracy: 0.8288 - auc_4: 0.9870 - val_loss: 0.8418 -
val_categorical_accuracy: 0.6772 - val_auc_4: 0.9490
Epoch 48/50
76/76 [==============================] - 23s 304ms/step - loss: 0.4522 -
categorical_accuracy: 0.8259 - auc_4: 0.9869 - val_loss: 0.8180 -
val_categorical_accuracy: 0.7378 - val_auc_4: 0.9557
Epoch 49/50
76/76 [==============================] - 23s 303ms/step - loss: 0.4365 -
categorical_accuracy: 0.8288 - auc_4: 0.9874 - val_loss: 1.0895 -
val_categorical_accuracy: 0.6639 - val_auc_4: 0.9303
Epoch 50/50
76/76 [==============================] - 23s 304ms/step - loss: 0.4269 -
categorical_accuracy: 0.8375 - auc_4: 0.9882 - val_loss: 0.8778 -
val_categorical_accuracy: 0.7187 - val_auc_4: 0.9485
```

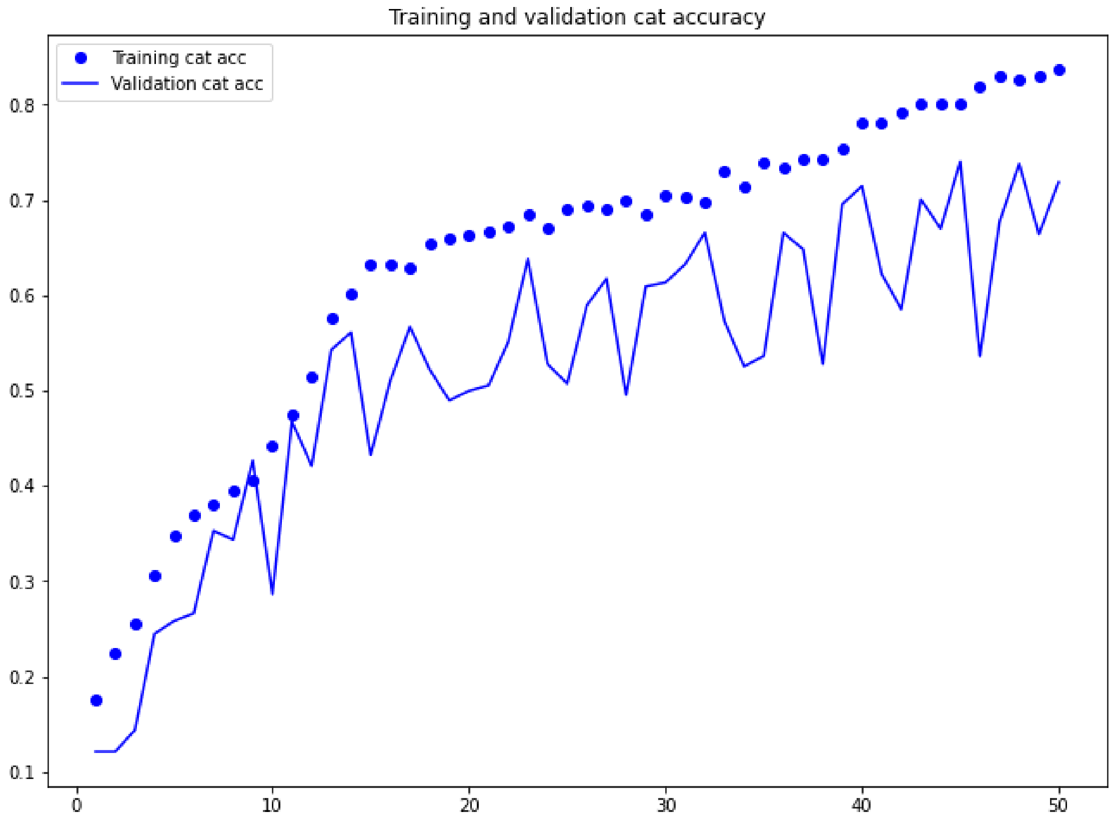
[56]:
```
mm1_dir = 'Data\\Models\\MM1'
MakeDirectory(mm1_dir)
with open(os.path.join(mm1_dir, "hsitory.json"), 'wb') as file_pi:
        pickle.dump(history_mm1.history, file_pi)
multimodal_model1.save(os.path.join(mm1_dir, "MM1.npy"))

#mm1_dir = 'Data\\Models\\CN1'
#history_mm1 = pickle.load(open(os.path.join(mm1_dir, "hsitory.json")), "rb")
#multimodal_model1 = load_model(os.path.join(mm1_dir, "MM1.npy"))
```

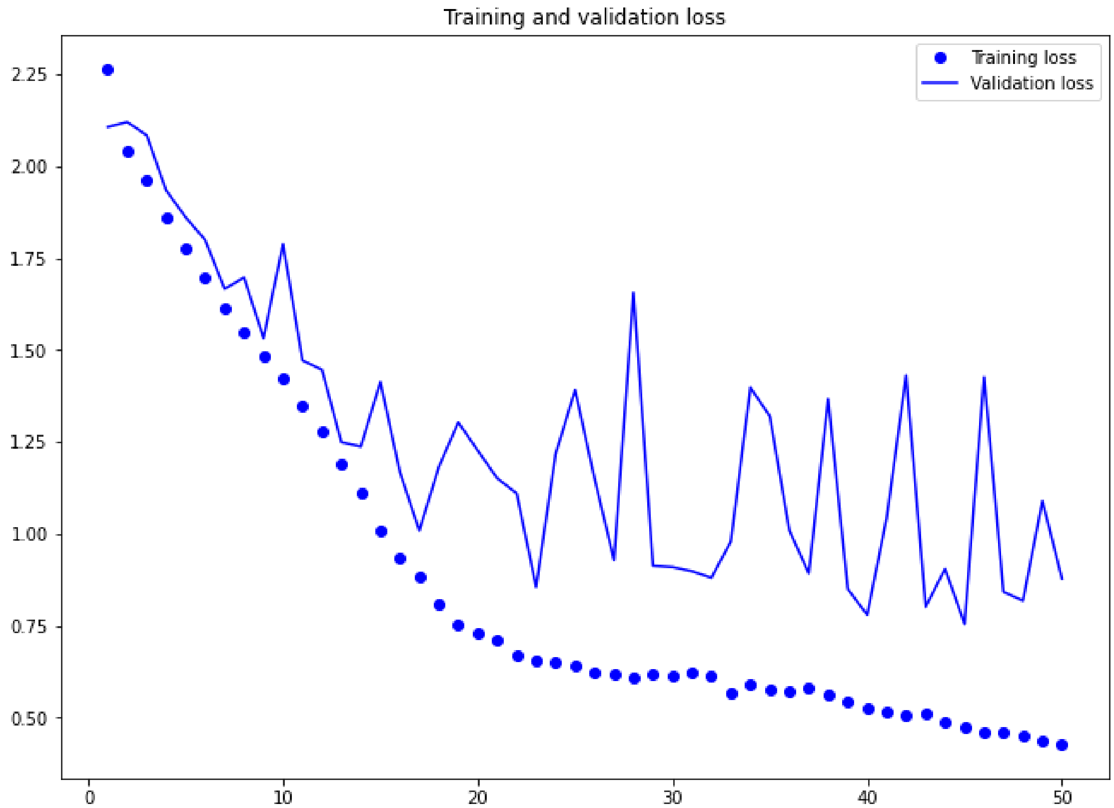
```
INFO:tensorflow:Assets written to: Data\Models\MM1\MM1.npy\assets
```

[203]:
```
# Code based on "F. Chollet, Deep learning with Python. 2021" work in Listing 5.
↪10 on page 137
acc = history_mm1.history['auc_4']
val_acc = history_mm1.history['val_auc_4']
cat_acc = history_mm1.history['categorical_accuracy']
val_cat_acc = history_mm1.history['val_categorical_accuracy']
loss = history_mm1.history['loss']
val_loss = history_mm1.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, cat_acc, 'bo', label='Training cat acc')
plt.plot(epochs, val_cat_acc, 'b', label='Validation cat acc')
plt.title('Training and validation cat accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.figure()

plt.show()
```

```
confusion_matrix_cnn(multimodal_model1, [rxti, x_train_vectors], y_train,␣
 ↪labels=EMOTIONAL_CATEGORIES, name="MM1 Train: ")
confusion_matrix_cnn(multimodal_model1, [rxvi, x_validate_vectors], y_validate,␣
 ↪labels=EMOTIONAL_CATEGORIES, name="MM1 Validate: ")
confusion_matrix_cnn(multimodal_model1, [rxei, x_test_vectors], y_test,␣
 ↪labels=EMOTIONAL_CATEGORIES, name="MM1 Test: ")
```

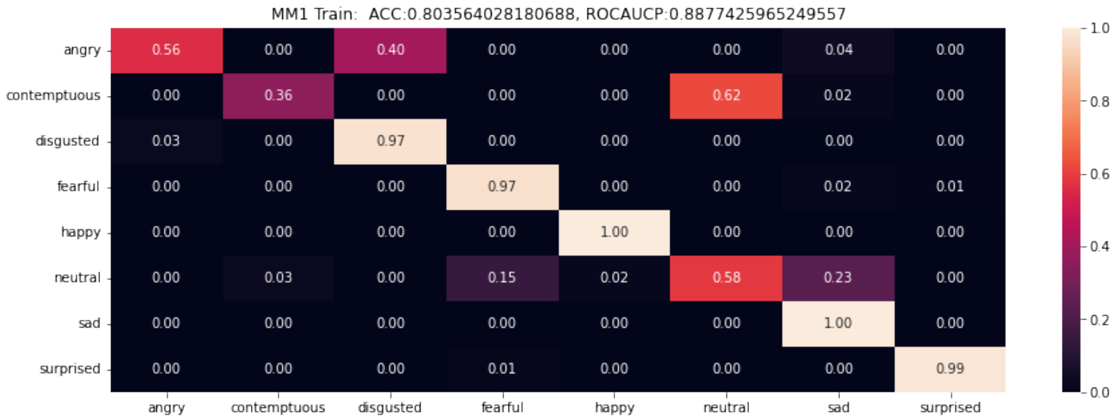


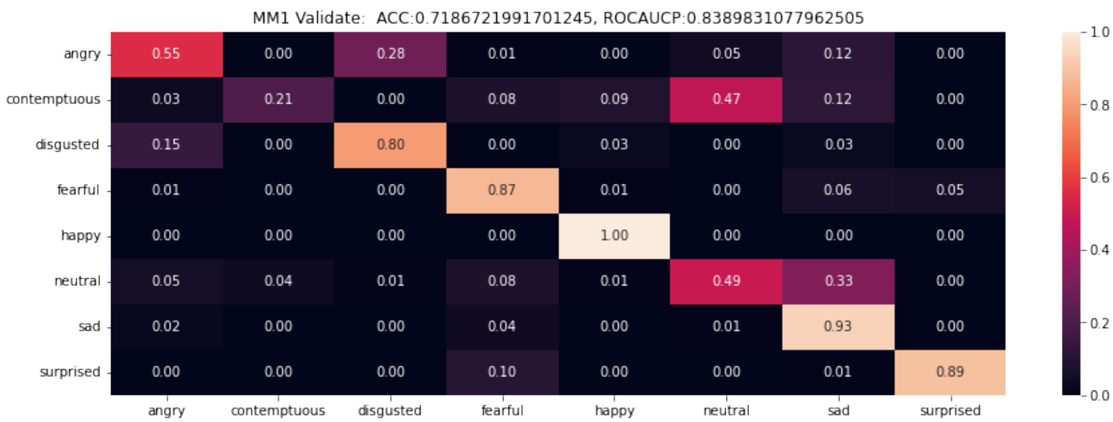

Training and validation cat accuracy
Training cat acc
Validation cat acc
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
10
20
30
40
50

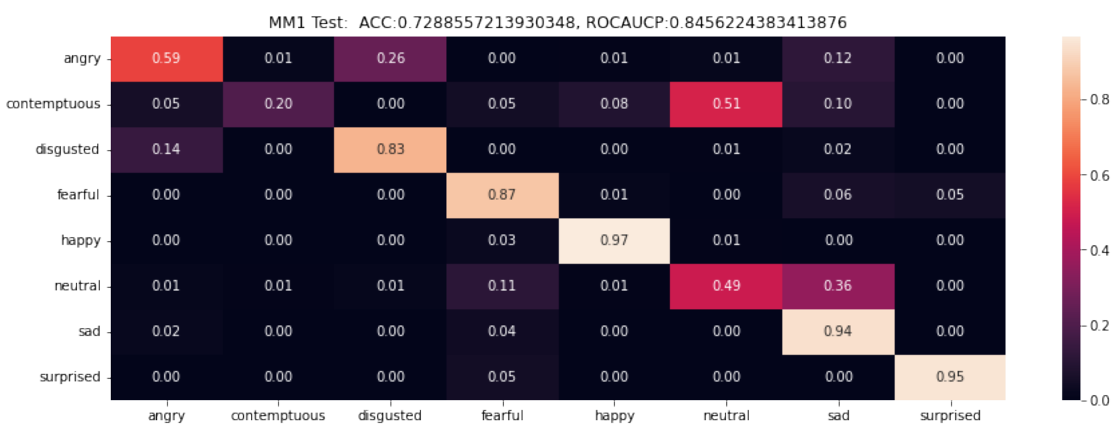



```
<Figure size 792x576 with 0 Axes>
```

```
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
```



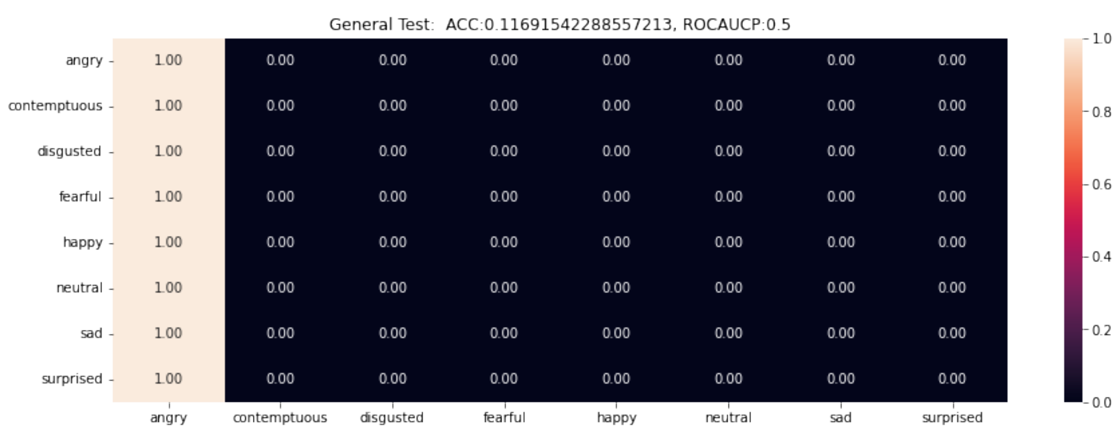


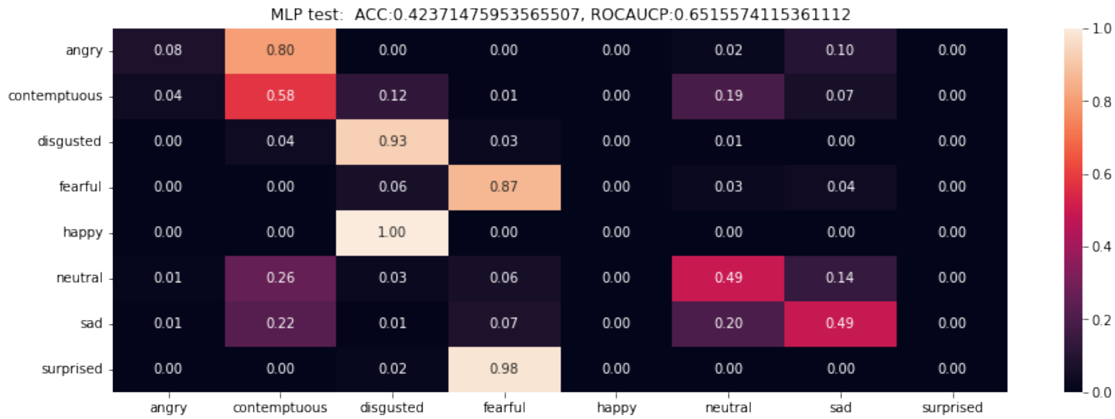


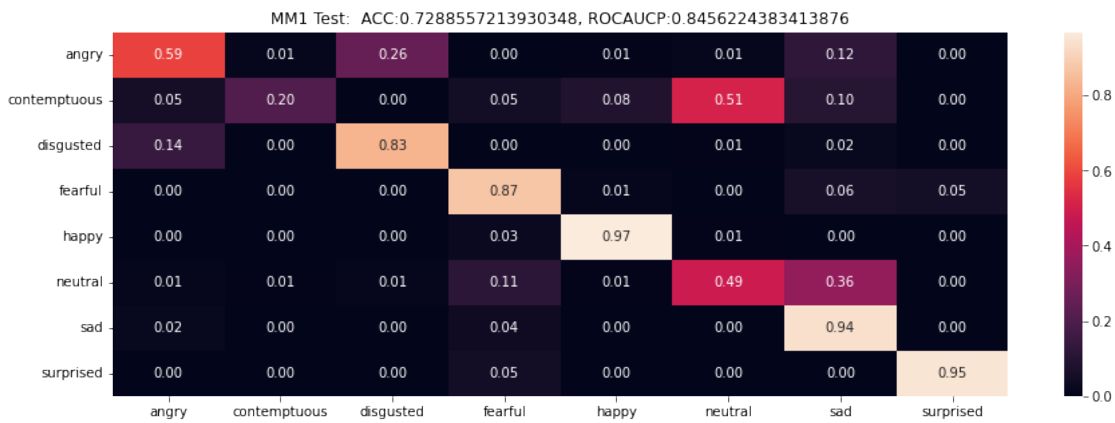
6. Report the results using the evaluation procedure that you argued for at the beginning of the lab. Results should be reported with proper statistical comparisons and proper visualizations.

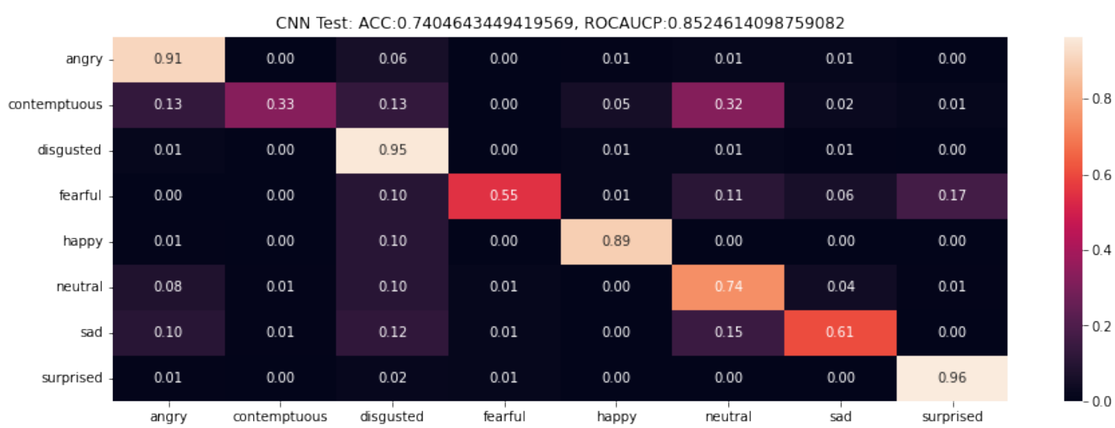
[204]:
```
confusion_matrix_cnn(generalized_mlp, X_test, y_test,␣
↪labels=EMOTIONAL_CATEGORIES, name="General Test: ")
confusion_matrix_cnn(vector_mlp, x_test_vectors, y_test,␣
↪labels=EMOTIONAL_CATEGORIES, name='MLP test: ')
confusion_matrix_cnn(multimodal_model1, [rxei, x_test_vectors], y_test,␣
↪labels=EMOTIONAL_CATEGORIES, name="MM1 Test: ")
confusion_matrix_cnn(conv_model, rxei, y_test, labels=EMOTIONAL_CATEGORIES,␣
↪name= "CNN Test:")
```

```
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
```









We are ordering the heat maps of test set predictions of each model based on the accuracy score.

The ordering is, General Vector and Image MLP model, Vector MLP model, Multimodal Vector and Image model,and finaly Image CNN model. As the accuracy increases, the more we guess correctly in the test set, the higher the values are in the main diagonal. This is represented with the colors white being 1.0 and dark purple being 0.0. Both the General and Vector MLP models cannot classify the Emotions from the image. With the multimodal, we begin to see the classification of most of the emotions, except for contemptuous and neutral. Interesting that neutral emotion would be difficult to classify, seeing that the vector differences of all landmarks would be 0. CNN was the best model, where it was able to distinguish the contemptuous and neutral emotions, but now falsely classifying an angry face with contemptuous, neutral, and sad.

1.6 Final Analysis

7. Finally, you have freedom to perform any other analysis you want. Please explain what analysis you are investigating and why.

From our original design, we used as input the (emote image, the Vector difference of the emote facial landmarks versus the neutral facial landmarks). Seeing how the Vector MLP was able to help the CNN emote images data, we would also like to add in the emote landmark positions. Like the image, the location may provide another way to classify the emotion and help the Multimodal model become general.

[214]:
```
cn = list()
for num in range(0, TOTAL_LANDMARKS*3):
    cn.append(f'P{num}')

df_ev = df_all_image_data
df_pos = pd.DataFrame(np_emote_landmark_data.reshape((TOTAL_PICS,␣
↪TOTAL_LANDMARKS*3)), columns= cn)
df_evp = df_ev.join(df_pos)
df_evp
```

[214]:
```
             0         1         2         3         4         5         6  \
0     0.905882  0.901961  0.894118  0.905882  0.901961  0.898039  0.905882
1     0.886275  0.882353  0.874510  0.886275  0.882353  0.874510  0.882353
2     0.901961  0.901961  0.894118  0.901961  0.901961  0.898039  0.905882
3     0.882353  0.878431  0.874510  0.882353  0.878431  0.874510  0.882353
4     0.882353  0.878431  0.878431  0.878431  0.874510  0.874510  0.882353
…          …         …         …         …         …         …         …
4819  0.886275  0.886275  0.894118  0.886275  0.886275  0.894118  0.890196
4820  0.890196  0.890196  0.901961  0.890196  0.894118  0.901961  0.898039
4821  0.874510  0.878431  0.886275  0.878431  0.882353  0.890196  0.882353
4822  0.882353  0.882353  0.890196  0.882353  0.886275  0.894118  0.886275
4823  0.878431  0.878431  0.886275  0.882353  0.882353  0.886275  0.882353

             7         8         9  …     P1394     P1395     P1396  \
0     0.905882  0.898039  0.905882  …  0.020225  0.774524  0.409666
1     0.882353  0.874510  0.886275  …  0.021096  0.777541  0.414701
2     0.905882  0.901961  0.905882  …  0.019768  0.772078  0.408849
```

```
3     0.878431  0.874510  0.882353  …  0.024333  0.772566  0.417264
4     0.874510  0.874510  0.882353  …  0.023972  0.773015  0.416928
…          …         …         …  …         …         …         …
4819  0.890196  0.894118  0.894118  … -0.026517  0.325018  0.419400
4820  0.898039  0.905882  0.898039  … -0.020713  0.325272  0.414747
4821  0.882353  0.894118  0.882353  … -0.022488  0.322666  0.408635
4822  0.886275  0.894118  0.890196  … -0.019810  0.318520  0.423897
4823  0.886275  0.886275  0.886275  … -0.019636  0.327205  0.417244

         P1397     P1398     P1399     P1400     P1401     P1402     P1403
0     0.008248  0.798379  0.392333  0.102728  0.799218  0.390161  0.109001
1     0.008410  0.804557  0.399260  0.099748  0.805672  0.397170  0.105581
2     0.007717  0.793189  0.392089  0.102869  0.793904  0.389965  0.109241
3     0.013341  0.809975  0.405085  0.105880  0.811142  0.400644  0.112569
4     0.013071  0.807879  0.404066  0.105915  0.808761  0.399656  0.112616
…          …         …         …         …         …         …         …
4819 -0.031397  0.427655  0.404330 -0.077694  0.439539  0.398927 -0.084478
4820 -0.026204  0.429772  0.401812 -0.068689  0.442248  0.396089 -0.075080
4821 -0.026588  0.433761  0.390086 -0.077396  0.447273  0.382953 -0.083736
4822 -0.023350  0.422027  0.401158 -0.079149  0.434996  0.392872 -0.085747
4823 -0.023436  0.432941  0.397934 -0.075095  0.446698  0.389891 -0.081156

[4824 rows x 35448 columns]
```

[225]:

```python
X_all = df_evp.to_numpy()
y_all = df_image_data["Emotion"].to_numpy()

label_encoder_all = LabelEncoder()
int_encoded_all = label_encoder.fit_transform(y_all)
oh_encoder_all = OneHotEncoder(sparse=False)
int_encoded_all = int_encoded.reshape(len(int_encoded), 1)
y_oh_all = oh_encoder.fit_transform(int_encoded)


X_train_all, X_test_all, y_train_all, y_test_all = train_test_split(X_all,␣
↪y_oh_all, test_size=0.25, random_state=123321)
X_train_all, X_validate_all, y_train_all, y_validate_all =␣
↪train_test_split(X_train_all, y_train_all, test_size=0.333,␣
↪random_state=123321, stratify=y_train_all)


print(f"X_train:{X_train_all.shape}")
print(f"y_train:{y_train_all.shape}")
print(f"X_validate:{X_validate_all.shape}")
print(f"y_validate:{y_validate_all.shape}")
print(f"X_test:{X_test_all.shape}")
print(f"y_test:{y_test_all.shape}")
```

```
X_train:(2413, 35448)
y_train:(2413, 8)
X_validate:(1205, 35448)
y_validate:(1205, 8)
X_test:(1206, 35448)
y_test:(1206, 8)
```

[226]:

```
#np_all_image_data = df_all_image_data.to_numpy()
#np_all_image_data[:,:new_flatten_img_list.shape[1]]

x_train_images_all = X_train_all[:,:new_flatten_img_list.shape[1]]
x_train_vectors_all  = X_train_all[:,(new_flatten_img_list.shape[1]+1):
 ↪(new_flatten_img_list.shape[1] + (TOTAL_LANDMARKS * 3) + 1)]
x_train_pos_all= X_train_all[:,(new_flatten_img_list.shape[1] +␣
 ↪(TOTAL_LANDMARKS * 3) + 2):]

x_validate_images_all = X_validate_all[:,:new_flatten_img_list.shape[1]]
x_validate_vectors_all = X_validate_all[:,(new_flatten_img_list.shape[1]+1):
 ↪(new_flatten_img_list.shape[1] + (TOTAL_LANDMARKS * 3) + 1)]
x_validate_pos_all = X_validate_all[:,(new_flatten_img_list.shape[1] +␣
 ↪(TOTAL_LANDMARKS * 3) + 2):]

x_test_images_all = X_test_all[:,:new_flatten_img_list.shape[1]]
x_test_vectors_all = X_test_all[:,(new_flatten_img_list.shape[1]+1):
 ↪(new_flatten_img_list.shape[1] + (TOTAL_LANDMARKS * 3) + 1)]
x_test_pos_all = X_test_all[:,(new_flatten_img_list.shape[1] + (TOTAL_LANDMARKS␣
 ↪* 3) + 2):]

rxti_all = x_train_images_all.reshape((x_train_images_all.shape[0], MIN_X,␣
 ↪MIN_Y, 3))
rxvi_all = x_validate_images_all.reshape((x_validate_images_all.shape[0],␣
 ↪MIN_X, MIN_Y, 3))
rxei_all = x_test_images_all.reshape((x_test_images_all.shape[0], MIN_X, MIN_Y,␣
 ↪3))

print(f"x_train_images_all:{x_train_images_all.shape}")
print(f"x_train_positions_all:{x_train_pos_all.shape}")
print(f"x_train_vectors_all:{x_train_vectors_all.shape}")
print(f"x_validate_images_all:{x_validate_images_all.shape}")
print(f"x_validate_positions_all:{x_validate_pos_all.shape}")
print(f"x_validate_vectors_all:{x_validate_vectors_all.shape}")
print(f"x_test_images_all:{x_test_images_all.shape}")
print(f"x_test_positions_all:{x_test_pos_all.shape}")
print(f"x_test_vectors_all:{x_test_vectors_all.shape}")
```

```
x_train_images_all:(2413, 32640)
x_train_positions_all:(2413, 1402)
```

```
x_train_vectors_all:(2413, 1404)
x_validate_images_all:(1205, 32640)
x_validate_positions_all:(1205, 1402)
x_validate_vectors_all:(1205, 1404)
x_test_images_all:(1206, 32640)
x_test_positions_all:(1206, 1402)
x_test_vectors_all:(1206, 1404)
```

[222]:
```
point_mlp = Sequential()
point_mlp.add(Dense(input_dim=x_train_pos_all.shape[1], units=991,
                            kernel_initializer= tf.keras.initializers.
 ↪GlorotUniform(),
                            kernel_regularizer= tf.keras.regularizers.
 ↪l1_l2(l1=1e-5, l2=1e-4),
                            bias_regularizer= tf.keras.regularizers.l2(1e-4),
                            activity_regularizer= tf.keras.regularizers.
 ↪l2(1e-5),
                            activation='relu',
                            name='input_pos_dense2053'))

point_mlp.add(Dense(units=381,
                            kernel_initializer=tf.keras.initializers.
 ↪GlorotUniform(),
                            kernel_regularizer= tf.keras.regularizers.
 ↪l1_l2(l1=1e-5, l2=1e-4),
                            bias_regularizer= tf.keras.regularizers.l2(1e-4),
                            activity_regularizer= tf.keras.regularizers.
 ↪l2(1e-5),
                            activation='relu',
                            name='mid_pos_dense991'))

point_mlp.add(Dense(units=109,
                            kernel_initializer=tf.keras.initializers.
 ↪GlorotUniform(),
                            kernel_regularizer= tf.keras.regularizers.
 ↪l1_l2(l1=1e-5, l2=1e-4),
                            bias_regularizer= tf.keras.regularizers.l2(1e-4),
                            activity_regularizer= tf.keras.regularizers.
 ↪l2(1e-5),
                            activation='relu',
                            name='mid_pos_dense381'))

point_mlp.add(Dense(units=31,
                            kernel_initializer=tf.keras.initializers.
 ↪GlorotUniform(),
                            kernel_regularizer= tf.keras.regularizers.
 ↪l1_l2(l1=1e-5, l2=1e-4),
```

```
                                bias_regularizer= tf.keras.regularizers.l2(1e-4),
                                activity_regularizer= tf.keras.regularizers.
 ↪l2(1e-5),
                                activation='relu',
                                name='mid_pos_dense109'))


point_mlp.add(Dense(units = len(EMOTIONAL_CATEGORIES),
                                kernel_regularizer= tf.keras.regularizers.
 ↪l1_l2(l1=1e-5, l2=1e-4),
                                bias_regularizer= tf.keras.regularizers.l2(1e-4),
                                activity_regularizer= tf.keras.regularizers.
 ↪l2(1e-5),
                                activation='sigmoid',
                                name='output_pos_layer'))

point_mlp.add(Activation('softmax'))


point_mlp.compile(loss='categorical_crossentropy',
                        optimizer= tf.keras.optimizers.RMSprop(learning_rate =␣
 ↪0.09973),
                        metrics=['accuracy', 'categorical_accuracy'])

#history_vector_mlp = vector_mlp.fit(x_train_vectors_all,
#                           y_train_all,
#                           batch_size = 32,
#                           epochs=100,
#                           shuffle=True,
#                           verbose=1,
#                           validation_data=(x_validate_vectors,y_validate),
#                           callbacks=[tf.keras.callbacks.
 ↪ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.
 ↪0001)])
#
#y_hat_vector_mlp = vector_mlp.predict(x_train_vectors_all)
```

[227]:

```
# Combining the two together
combined_input_2 = tf.keras.layers.concatenate([conv_model.output,vector_mlp.
 ↪output, point_mlp.output])
mm2_out = Dense(24, activation="relu")(combined_input_2)
mm2_out = Dense(8, activation="softmax")(mm2_out)

multimodal_model2 = tf.keras.models.Model(inputs= [conv_model.input,vector_mlp.
 ↪input, point_mlp.input], outputs= mm2_out)
```

```
multimodal_model2.compile(optimizer="rmsprop", loss="categorical_crossentropy",␣
↪metrics=[
    tf.keras.metrics.CategoricalAccuracy(),
    tf.keras.metrics.AUC()
])

history_mm2 = multimodal_model2.fit(x=[rxti_all, x_train_vectors_all,␣
↪x_train_pos_all], y=y_train_all,
                                    validation_data=([rxvi_all,␣
↪x_validate_vectors_all, x_validate_pos_all], y_validate_all),
                                    epochs=50,
                                    batch_size=32)
```

```
Epoch 1/50

C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(

76/76 [==============================] - 43s 559ms/step - loss: 2.4112 -
categorical_accuracy: 0.1604 - auc_12: 0.5546 - val_loss: 2.1212 -
val_categorical_accuracy: 0.1519 - val_auc_12: 0.5294
Epoch 2/50
76/76 [==============================] - 44s 581ms/step - loss: 1.9825 -
categorical_accuracy: 0.2967 - auc_12: 0.7382 - val_loss: 2.0725 -
val_categorical_accuracy: 0.1444 - val_auc_12: 0.5834
Epoch 3/50
76/76 [==============================] - 42s 559ms/step - loss: 1.7946 -
categorical_accuracy: 0.3780 - auc_12: 0.8290 - val_loss: 1.9819 -
val_categorical_accuracy: 0.1900 - val_auc_12: 0.7050
Epoch 4/50
76/76 [==============================] - 43s 562ms/step - loss: 1.6011 -
categorical_accuracy: 0.4306 - auc_12: 0.8665 - val_loss: 1.9027 -
val_categorical_accuracy: 0.1751 - val_auc_12: 0.6943
Epoch 5/50
76/76 [==============================] - 43s 568ms/step - loss: 1.4397 -
categorical_accuracy: 0.4538 - auc_12: 0.8908 - val_loss: 1.6563 -
val_categorical_accuracy: 0.2647 - val_auc_12: 0.8113
Epoch 6/50
76/76 [==============================] - 43s 564ms/step - loss: 1.3081 -
categorical_accuracy: 0.5093 - auc_12: 0.9083 - val_loss: 1.5576 -
val_categorical_accuracy: 0.3228 - val_auc_12: 0.8253
Epoch 7/50
76/76 [==============================] - 43s 561ms/step - loss: 1.1882 -
categorical_accuracy: 0.5396 - auc_12: 0.9239 - val_loss: 1.6058 -
```

```
val_categorical_accuracy: 0.3494 - val_auc_12: 0.8143
Epoch 8/50
76/76 [==============================] - 45s 588ms/step - loss: 1.0989 -
categorical_accuracy: 0.5470 - auc_12: 0.9287 - val_loss: 1.5601 -
val_categorical_accuracy: 0.3444 - val_auc_12: 0.8246
Epoch 9/50
76/76 [==============================] - 44s 575ms/step - loss: 1.0268 -
categorical_accuracy: 0.5806 - auc_12: 0.9342 - val_loss: 1.2130 -
val_categorical_accuracy: 0.4564 - val_auc_12: 0.8987
Epoch 10/50
76/76 [==============================] - 44s 585ms/step - loss: 0.9608 -
categorical_accuracy: 0.5810 - auc_12: 0.9396 - val_loss: 1.3702 -
val_categorical_accuracy: 0.4332 - val_auc_12: 0.8707
Epoch 11/50
76/76 [==============================] - 44s 583ms/step - loss: 0.8973 -
categorical_accuracy: 0.6005 - auc_12: 0.9466 - val_loss: 1.2766 -
val_categorical_accuracy: 0.4846 - val_auc_12: 0.8909
Epoch 12/50
76/76 [==============================] - 44s 579ms/step - loss: 0.8485 -
categorical_accuracy: 0.6386 - auc_12: 0.9514 - val_loss: 0.9950 -
val_categorical_accuracy: 0.5585 - val_auc_12: 0.9300
Epoch 13/50
76/76 [==============================] - 44s 574ms/step - loss: 0.8272 -
categorical_accuracy: 0.6183 - auc_12: 0.9516 - val_loss: 1.4200 -
val_categorical_accuracy: 0.4581 - val_auc_12: 0.8758
Epoch 14/50
76/76 [==============================] - 44s 577ms/step - loss: 0.7873 -
categorical_accuracy: 0.6366 - auc_12: 0.9556 - val_loss: 1.7437 -
val_categorical_accuracy: 0.3809 - val_auc_12: 0.8252
Epoch 15/50
76/76 [==============================] - 44s 579ms/step - loss: 0.7505 -
categorical_accuracy: 0.6436 - auc_12: 0.9592 - val_loss: 0.9679 -
val_categorical_accuracy: 0.5485 - val_auc_12: 0.9324
Epoch 16/50
76/76 [==============================] - 43s 567ms/step - loss: 0.7455 -
categorical_accuracy: 0.6482 - auc_12: 0.9597 - val_loss: 1.1600 -
val_categorical_accuracy: 0.5585 - val_auc_12: 0.9171
Epoch 17/50
76/76 [==============================] - 43s 564ms/step - loss: 0.7102 -
categorical_accuracy: 0.6693 - auc_12: 0.9630 - val_loss: 1.3220 -
val_categorical_accuracy: 0.5411 - val_auc_12: 0.9026
Epoch 18/50
76/76 [==============================] - 43s 571ms/step - loss: 0.6759 -
categorical_accuracy: 0.6884 - auc_12: 0.9664 - val_loss: 1.4485 -
val_categorical_accuracy: 0.4780 - val_auc_12: 0.8721
Epoch 19/50
76/76 [==============================] - 43s 570ms/step - loss: 0.6779 -
categorical_accuracy: 0.6937 - auc_12: 0.9669 - val_loss: 1.3673 -
```

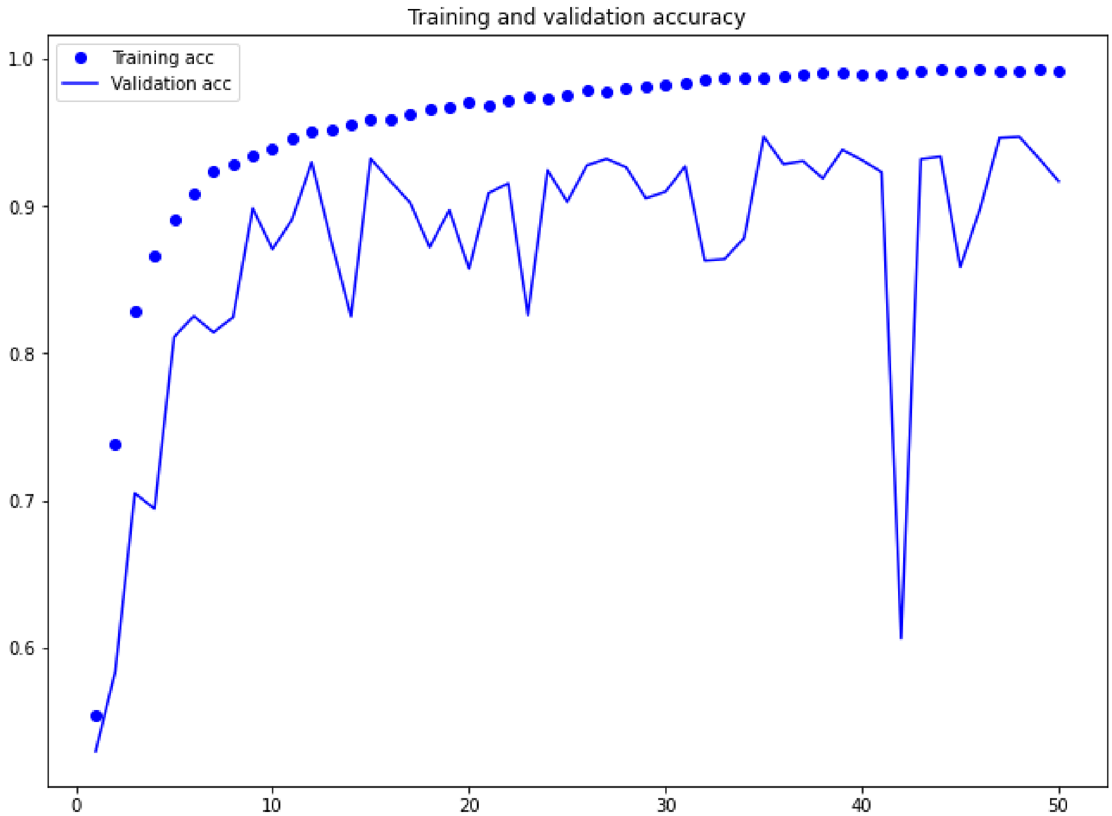
```
val_categorical_accuracy: 0.5228 - val_auc_12: 0.8974
Epoch 20/50
76/76 [==============================] - 44s 579ms/step - loss: 0.6358 -
categorical_accuracy: 0.7033 - auc_12: 0.9702 - val_loss: 1.6245 -
val_categorical_accuracy: 0.4830 - val_auc_12: 0.8576
Epoch 21/50
76/76 [==============================] - 43s 571ms/step - loss: 0.6527 -
categorical_accuracy: 0.6983 - auc_12: 0.9688 - val_loss: 1.1916 -
val_categorical_accuracy: 0.5444 - val_auc_12: 0.9090
Epoch 22/50
76/76 [==============================] - 44s 579ms/step - loss: 0.6267 -
categorical_accuracy: 0.7074 - auc_12: 0.9714 - val_loss: 1.2128 -
val_categorical_accuracy: 0.5793 - val_auc_12: 0.9155
Epoch 23/50
76/76 [==============================] - 44s 582ms/step - loss: 0.5995 -
categorical_accuracy: 0.7339 - auc_12: 0.9743 - val_loss: 1.8956 -
val_categorical_accuracy: 0.4216 - val_auc_12: 0.8261
Epoch 24/50
76/76 [==============================] - 44s 586ms/step - loss: 0.6097 -
categorical_accuracy: 0.7232 - auc_12: 0.9729 - val_loss: 1.0916 -
val_categorical_accuracy: 0.5917 - val_auc_12: 0.9246
Epoch 25/50
76/76 [==============================] - 46s 602ms/step - loss: 0.5845 -
categorical_accuracy: 0.7360 - auc_12: 0.9753 - val_loss: 1.3161 -
val_categorical_accuracy: 0.5817 - val_auc_12: 0.9029
Epoch 26/50
76/76 [==============================] - 43s 570ms/step - loss: 0.5417 -
categorical_accuracy: 0.7559 - auc_12: 0.9792 - val_loss: 1.0423 -
val_categorical_accuracy: 0.5751 - val_auc_12: 0.9278
Epoch 27/50
76/76 [==============================] - 43s 564ms/step - loss: 0.5559 -
categorical_accuracy: 0.7509 - auc_12: 0.9775 - val_loss: 1.0379 -
val_categorical_accuracy: 0.6299 - val_auc_12: 0.9323
Epoch 28/50
76/76 [==============================] - 43s 567ms/step - loss: 0.5291 -
categorical_accuracy: 0.7646 - auc_12: 0.9801 - val_loss: 1.0859 -
val_categorical_accuracy: 0.6340 - val_auc_12: 0.9265
Epoch 29/50
76/76 [==============================] - 45s 588ms/step - loss: 0.5122 -
categorical_accuracy: 0.7795 - auc_12: 0.9815 - val_loss: 1.2696 -
val_categorical_accuracy: 0.6075 - val_auc_12: 0.9055
Epoch 30/50
76/76 [==============================] - 43s 568ms/step - loss: 0.5080 -
categorical_accuracy: 0.7845 - auc_12: 0.9822 - val_loss: 1.2300 -
val_categorical_accuracy: 0.6008 - val_auc_12: 0.9099
Epoch 31/50
76/76 [==============================] - 43s 563ms/step - loss: 0.4817 -
categorical_accuracy: 0.8002 - auc_12: 0.9837 - val_loss: 1.1023 -
```

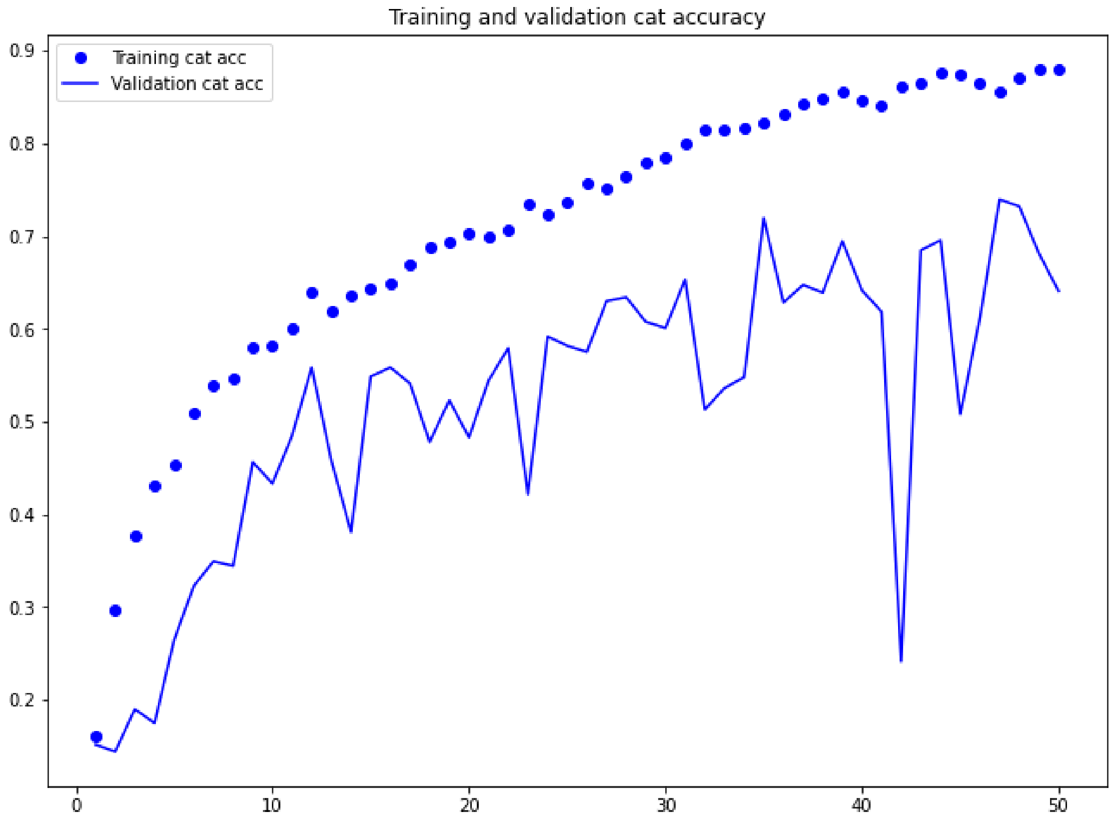
```
val_categorical_accuracy: 0.6531 - val_auc_12: 0.9272
Epoch 32/50
76/76 [==============================] - 43s 564ms/step - loss: 0.4614 -
categorical_accuracy: 0.8148 - auc_12: 0.9858 - val_loss: 1.7511 -
val_categorical_accuracy: 0.5129 - val_auc_12: 0.8631
Epoch 33/50
76/76 [==============================] - 43s 565ms/step - loss: 0.4377 -
categorical_accuracy: 0.8152 - auc_12: 0.9872 - val_loss: 1.7911 -
val_categorical_accuracy: 0.5361 - val_auc_12: 0.8641
Epoch 34/50
76/76 [==============================] - 42s 558ms/step - loss: 0.4381 -
categorical_accuracy: 0.8156 - auc_12: 0.9868 - val_loss: 1.5826 -
val_categorical_accuracy: 0.5477 - val_auc_12: 0.8783
Epoch 35/50
76/76 [==============================] - 43s 561ms/step - loss: 0.4374 -
categorical_accuracy: 0.8226 - auc_12: 0.9868 - val_loss: 0.8690 -
val_categorical_accuracy: 0.7195 - val_auc_12: 0.9473
Epoch 36/50
76/76 [==============================] - 43s 569ms/step - loss: 0.4114 -
categorical_accuracy: 0.8305 - auc_12: 0.9884 - val_loss: 1.1040 -
val_categorical_accuracy: 0.6282 - val_auc_12: 0.9287
Epoch 37/50
76/76 [==============================] - 44s 579ms/step - loss: 0.4006 -
categorical_accuracy: 0.8421 - auc_12: 0.9890 - val_loss: 1.0661 -
val_categorical_accuracy: 0.6473 - val_auc_12: 0.9307
Epoch 38/50
76/76 [==============================] - 44s 576ms/step - loss: 0.3795 -
categorical_accuracy: 0.8479 - auc_12: 0.9906 - val_loss: 1.2218 -
val_categorical_accuracy: 0.6390 - val_auc_12: 0.9189
Epoch 39/50
76/76 [==============================] - 44s 577ms/step - loss: 0.3731 -
categorical_accuracy: 0.8558 - auc_12: 0.9910 - val_loss: 0.9833 -
val_categorical_accuracy: 0.6946 - val_auc_12: 0.9384
Epoch 40/50
76/76 [==============================] - 43s 566ms/step - loss: 0.3818 -
categorical_accuracy: 0.8467 - auc_12: 0.9901 - val_loss: 1.1336 -
val_categorical_accuracy: 0.6415 - val_auc_12: 0.9314
Epoch 41/50
76/76 [==============================] - 43s 564ms/step - loss: 0.3855 -
categorical_accuracy: 0.8400 - auc_12: 0.9899 - val_loss: 1.1607 -
val_categorical_accuracy: 0.6183 - val_auc_12: 0.9232
Epoch 42/50
76/76 [==============================] - 43s 563ms/step - loss: 0.3701 -
categorical_accuracy: 0.8612 - auc_12: 0.9909 - val_loss: 5.0605 -
val_categorical_accuracy: 0.2415 - val_auc_12: 0.6062
Epoch 43/50
76/76 [==============================] - 43s 565ms/step - loss: 0.3508 -
categorical_accuracy: 0.8649 - auc_12: 0.9914 - val_loss: 1.0738 -
```

```
val_categorical_accuracy: 0.6846 - val_auc_12: 0.9321
Epoch 44/50
76/76 [==============================] - 42s 557ms/step - loss: 0.3337 -
categorical_accuracy: 0.8753 - auc_12: 0.9928 - val_loss: 1.0454 -
val_categorical_accuracy: 0.6954 - val_auc_12: 0.9338
Epoch 45/50
76/76 [==============================] - 40s 524ms/step - loss: 0.3468 -
categorical_accuracy: 0.8732 - auc_12: 0.9917 - val_loss: 1.9753 -
val_categorical_accuracy: 0.5079 - val_auc_12: 0.8587
Epoch 46/50
76/76 [==============================] - 40s 529ms/step - loss: 0.3341 -
categorical_accuracy: 0.8649 - auc_12: 0.9926 - val_loss: 1.4419 -
val_categorical_accuracy: 0.6116 - val_auc_12: 0.8979
Epoch 47/50
76/76 [==============================] - 41s 536ms/step - loss: 0.3487 -
categorical_accuracy: 0.8558 - auc_12: 0.9919 - val_loss: 0.9398 -
val_categorical_accuracy: 0.7394 - val_auc_12: 0.9465
Epoch 48/50
76/76 [==============================] - 40s 521ms/step - loss: 0.3335 -
categorical_accuracy: 0.8695 - auc_12: 0.9925 - val_loss: 0.9191 -
val_categorical_accuracy: 0.7320 - val_auc_12: 0.9473
Epoch 49/50
76/76 [==============================] - 41s 536ms/step - loss: 0.3271 -
categorical_accuracy: 0.8802 - auc_12: 0.9935 - val_loss: 1.1281 -
val_categorical_accuracy: 0.6813 - val_auc_12: 0.9326
Epoch 50/50
76/76 [==============================] - 40s 527ms/step - loss: 0.3430 -
categorical_accuracy: 0.8790 - auc_12: 0.9919 - val_loss: 1.3055 -
val_categorical_accuracy: 0.6407 - val_auc_12: 0.9171
```

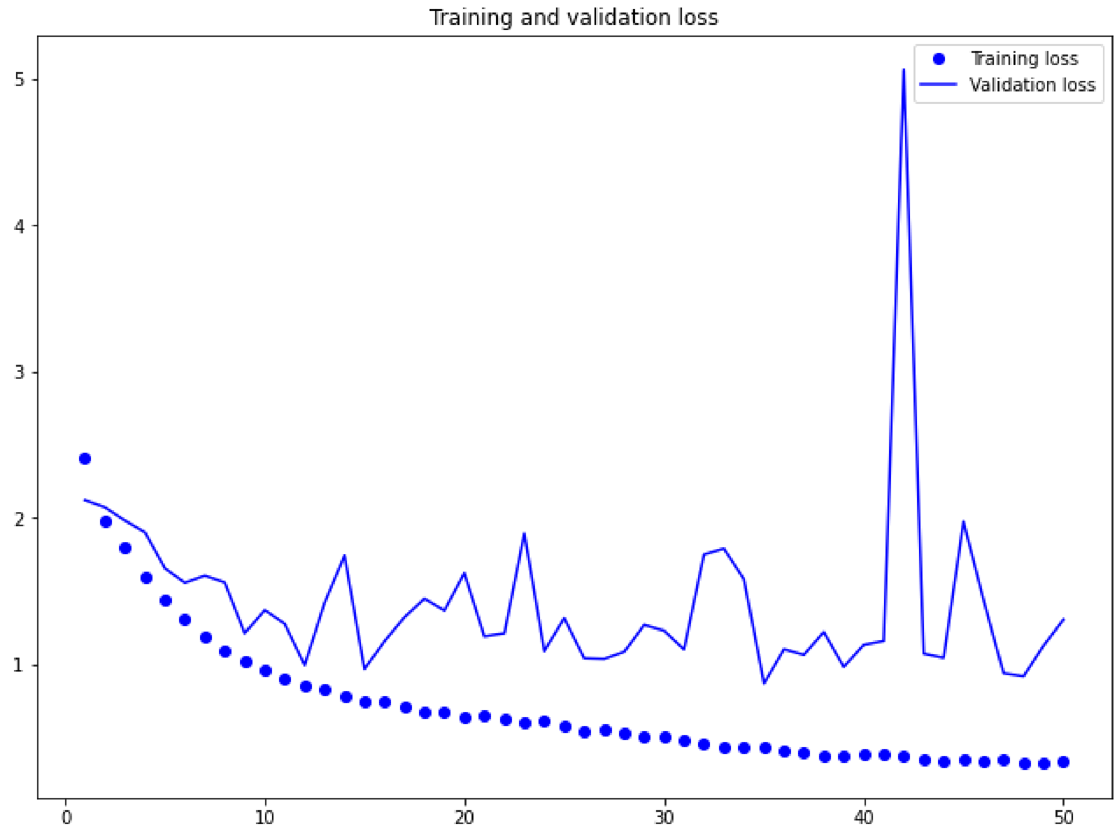
[244]:
```
# Code based on "F. Chollet, Deep learning with Python. 2021" work in Listing 5.
↪10 on page 137
acc = history_mm2.history['auc_12']
val_acc = history_mm2.history['val_auc_12']
cat_acc = history_mm2.history['categorical_accuracy']
val_cat_acc = history_mm2.history['val_categorical_accuracy']
loss = history_mm2.history['loss']
val_loss = history_mm2.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
```

```python
plt.plot(epochs, cat_acc, 'bo', label='Training cat acc')
plt.plot(epochs, val_cat_acc, 'b', label='Validation cat acc')
plt.title('Training and validation cat accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.figure()

plt.show()


confusion_matrix_cnn(multimodal_model2, [rxti_all, x_train_vectors_all,␣
 ↪x_train_pos_all], y_train_all, labels=EMOTIONAL_CATEGORIES, name="MM2 Train:␣
 ↪")
confusion_matrix_cnn(multimodal_model2, [rxvi_all, x_validate_vectors_all,␣
 ↪x_validate_pos_all], y_validate_all, labels=EMOTIONAL_CATEGORIES, name="MM2␣
 ↪Validate: ")
confusion_matrix_cnn(multimodal_model2, [rxei_all, x_test_vectors_all,␣
 ↪x_test_pos_all], y_test_all, labels=EMOTIONAL_CATEGORIES, name="MM2 Test: ")
```


Training and validation accuracy
Training acc
Validation acc
1.0
0.9
0.8
0.7
0.6
0
10
20
30
40
50

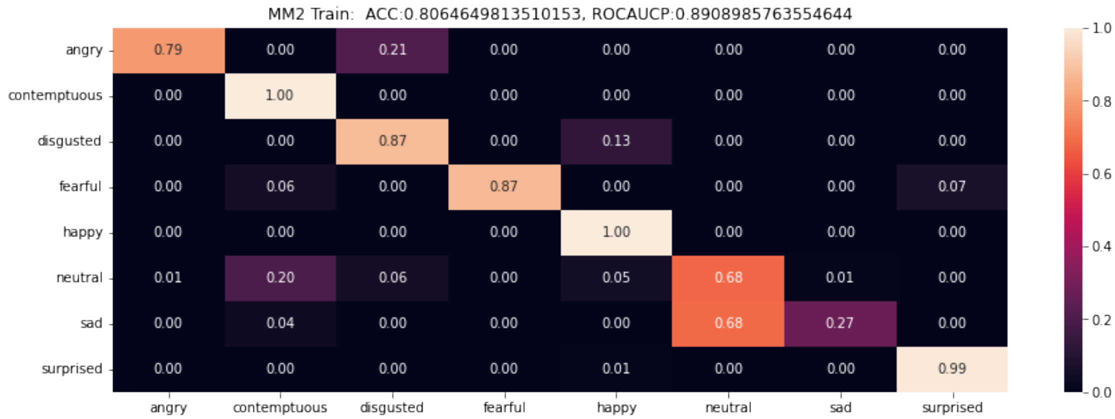


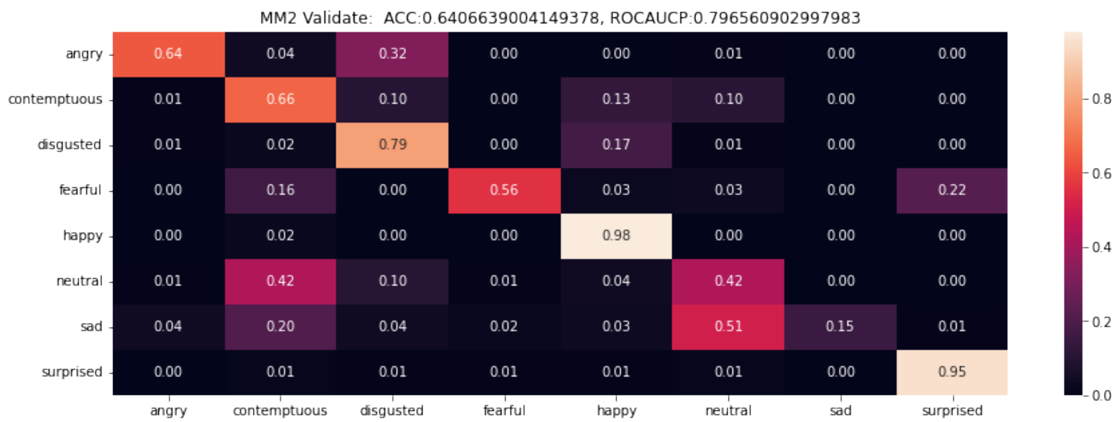
Training and validation cat accuracy
Training cat acc
Validation cat acc
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50

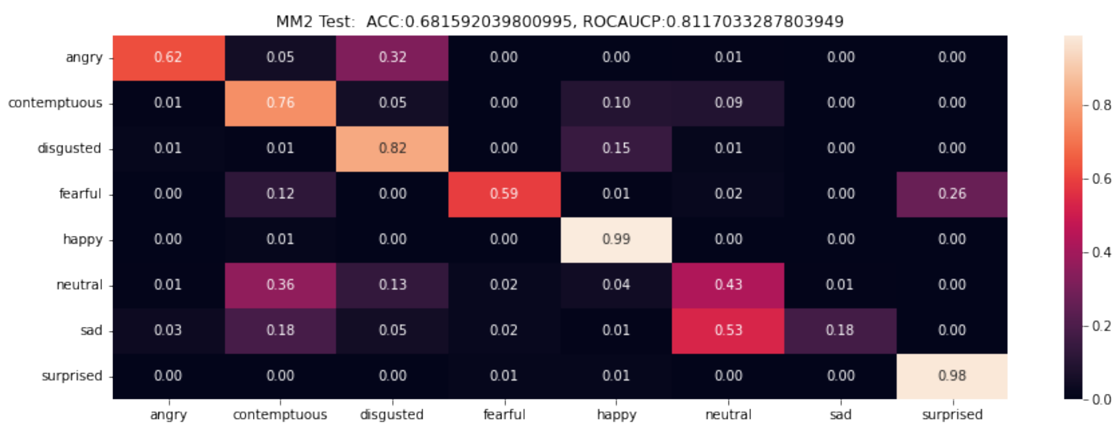



```
<Figure size 792x576 with 0 Axes>
```

```
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
C:\Users\Jake Klinkert\AppData\Local\Programs\Python\Python310\lib\site-
packages\tensorflow\python\data\ops\structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
```


MM2 Train: ACC:0.8064649813510153, ROCAUCP:0.8908985763554644
angry 0.79 0.00 0.21 0.00 0.00 0.00 0.00 0.00
contemptuous 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00
disgusted 0.00 0.00 0.87 0.00 0.13 0.00 0.00 0.00
fearful 0.00 0.06 0.00 0.87 0.00 0.00 0.00 0.07
happy 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00
neutral 0.01 0.20 0.06 0.00 0.05 0.68 0.01 0.00
sad 0.00 0.04 0.00 0.00 0.00 0.68 0.27 0.00
surprised 0.00 0.00 0.00 0.00 0.01 0.00 0.00 0.99
angry contemptuous disgusted fearful happy neutral sad surprised
1.0 0.8 0.6 0.4 0.2 0.0


MM2 Validate: ACC:0.6406639004149378, ROCAUCP:0.796560902997983
angry 0.64 0.04 0.32 0.00 0.00 0.01 0.00 0.00
contemptuous 0.01 0.66 0.10 0.00 0.13 0.10 0.00 0.00
disgusted 0.01 0.02 0.79 0.00 0.17 0.01 0.00 0.00
fearful 0.00 0.16 0.00 0.56 0.03 0.03 0.00 0.22
happy 0.00 0.02 0.00 0.00 0.98 0.00 0.00 0.00
neutral 0.01 0.42 0.10 0.01 0.04 0.42 0.00 0.00
sad 0.04 0.20 0.04 0.02 0.03 0.51 0.15 0.01
surprised 0.00 0.01 0.01 0.01 0.01 0.01 0.00 0.95
angry contemptuous disgusted fearful happy neutral sad surprised
0.8 0.6 0.4 0.2 0.0


MM2 Test: ACC:0.681592039800995, ROCAUCP:0.8117033287803949
angry 0.62 0.05 0.32 0.00 0.00 0.01 0.00 0.00
contemptuous 0.01 0.76 0.05 0.00 0.10 0.09 0.00 0.00
disgusted 0.01 0.01 0.82 0.00 0.15 0.01 0.00 0.00
fearful 0.00 0.12 0.00 0.59 0.01 0.02 0.00 0.26
happy 0.00 0.01 0.00 0.00 0.99 0.00 0.00 0.00
neutral 0.01 0.36 0.13 0.02 0.04 0.43 0.01 0.00
sad 0.03 0.18 0.05 0.02 0.01 0.53 0.18 0.00
surprised 0.00 0.00 0.00 0.01 0.01 0.00 0.00 0.98
angry contemptuous disgusted fearful happy neutral sad surprised
0.8 0.6 0.4 0.2 0.0

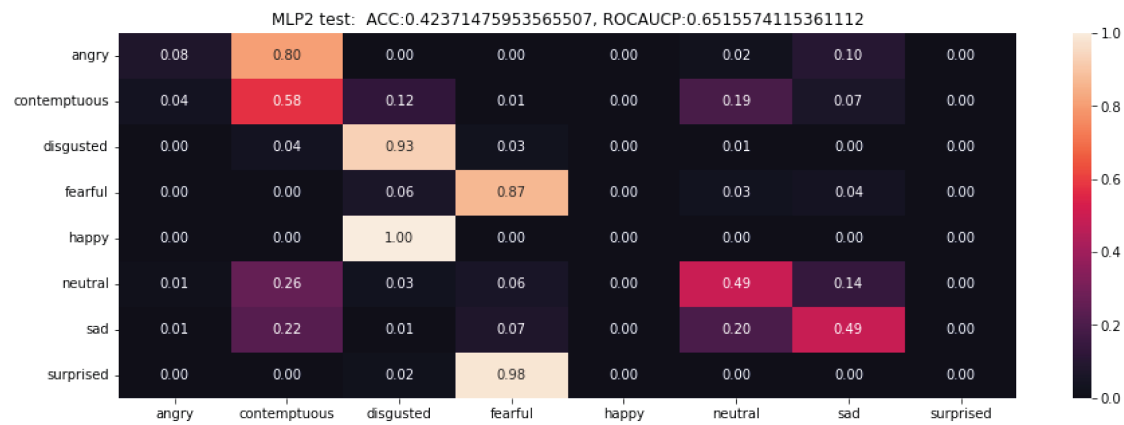
1.6.1 Comparison of models

[239]:
```
Image.open("Data/gm1_test.PNG")
```

[239]:

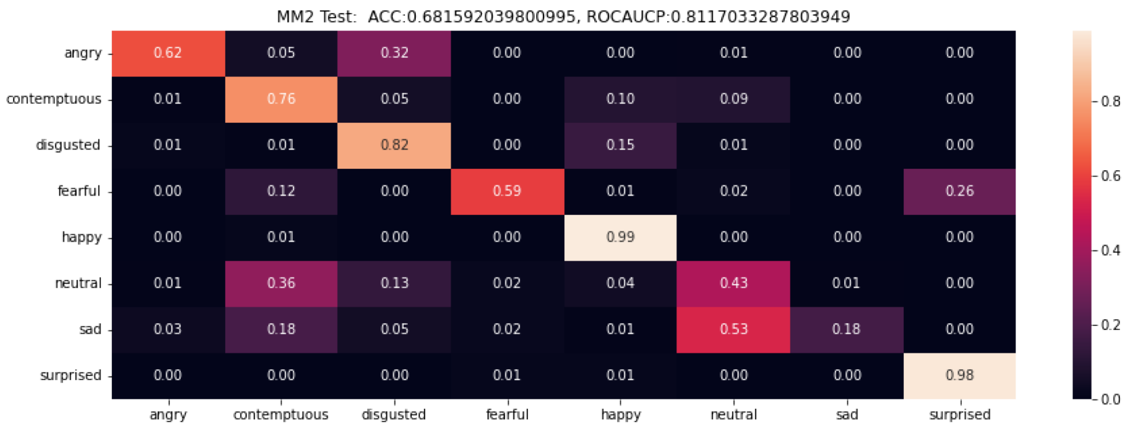
[240]:
```
Image.open("Data/mlp2_test.PNG")
```

[240]:



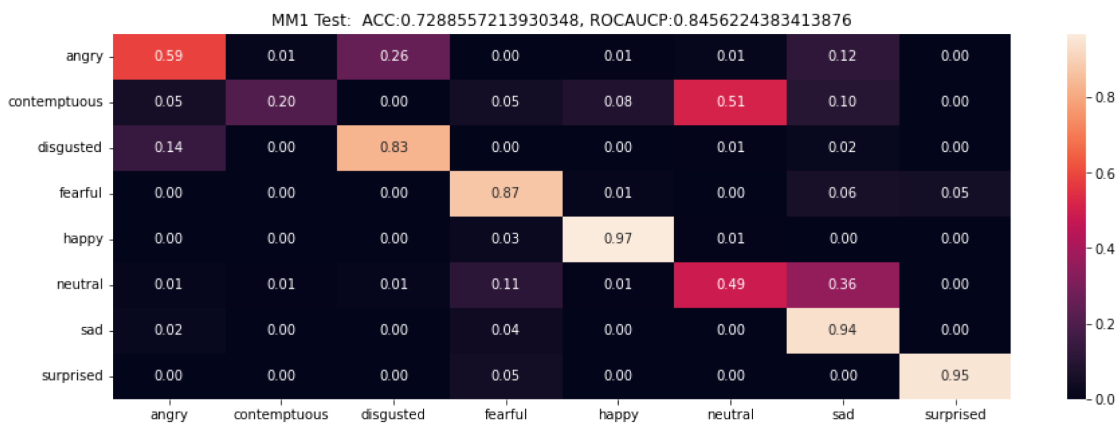
[245]:
```
Image.open("Data/mm2_test.PNG")
```

[245]:



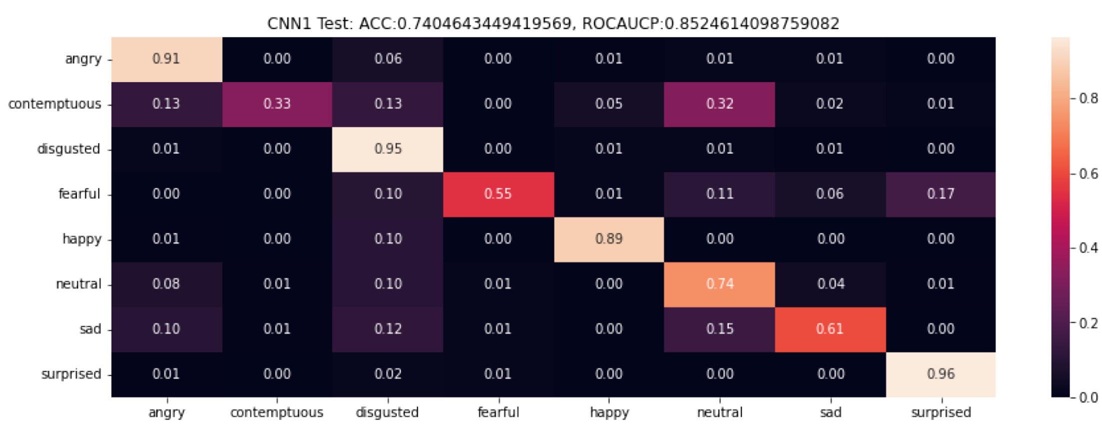
[246]:
```
Image.open("Data/mm1_test.PNG")
```

[246]:



[247]:
```
Image.open("Data/cnn1_test.PNG")
```

[247]:



Using the test sets to determine categorical accuracy and ROC AUC score, we can see that CNN has the best accuracy and ROC AUC score with our current experimental setup. Interestingly, we can see that Multimodal 2 distinguished the emotional faces better than Multimodal 1, but at the loss of generalization. Notice in Multimodal 2 we have more categories: false positives and false negatives. Compared to Multimodal 2 to Multimodal 1, the percentage of FP and FN are considerably lower but more spread out to other emotions. Lastly, the learning curve of the second Multimodal shows that it is still increasing, and the loss is still fluctuating, meaning that the model could be trained longer without overfitting the data.

1.7 Refrences

[1] O. Langner, R. Dotsch, G. Bijlstra, D. H. J. Wigboldus, S. T. Hawk, and A. van Knippenberg, "Presentation and validation of the Radboud Faces Database," Cognition and Emotion, vol. 24, no. 8, pp. 1377–1388, Dec. 2010, doi: 10.1080/02699930903485076.

[2] F. Chollet, Deep learning with Python. 2021.