In [1]:
```
import numpy as np
import pandas as pd
from numpy import asarray

import json
import cv2
from PIL import Image
import matplotlib.pyplot as plt

import os
import sys
from pathlib import Path
sys.path.append('../src')
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# importing model libraries
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from tensorflow.keras.layers import Dense, Activation, Dropout, Flatten, BatchNormalizat
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.metrics import categorical_accuracy
from tensorflow.keras.models import model_from_json,load_model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger, EarlyStopping, Reduce
from tensorflow.keras.optimizers import *
import tensorflow.keras.backend as K
face_cascade = cv2.CascadeClassifier('../src/haarcascade_frontalface_default.xml')
```

Save

In [2]:
```
inp1 = Path.home()/'Iron'/'inp1'
df = pd.read_csv(inp1/'Fer.csv', encoding = "ISO-8859-1")
df.head()
```

Save

Out[2]:

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

In [3]:
```
emo = {0:'other', 1:'other', 2:'other', 3:'happy', 4:'other', 5:'other', 6:'other'}
df['emo'] = df.emotion.map(emo).to_numpy()
df = pd.get_dummies(df, columns=['emo'])
df['happy_other'] = df[['emo_happy','emo_other']].apply(lambda x: pd.Series([x.values]),
df.head()
```

Save

Out[3]:

	emotion	pixels	Usage	emo_happy	emo_other	happy_other
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training	0	1	[0, 1]
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training	0	1	[0, 1]
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training	0	1	[0, 1]
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training	0	1	[0, 1]

	emotion	pixels	Usage	emo_happy	emo_other	happy_other
4	6	4 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training	0	1	[0, 1]

In [6]:
```
df['pixels1'] = [[float(x) for x in each.split()] for each in df['pixels']]
df['pixels2'] = df['pixels1'].apply(lambda x: np.asarray(x).reshape(48,48)).apply(lambda
df['pixels3'] = df['pixels2'].apply(lambda x: np.array([[[c] for c in i] for i in x]))
```

In [7]:
```
drop = ['emotion', 'Usage', 'pixels1', 'pixels2']
df.drop(drop, axis=1, inplace=True)
df.head()
```

Out[7]:

	pixels	emo_happy	emo_other	happy_other	pixels3
0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	0	1	[0, 1]	[[[70.0], [80.0], [82.0], [72.0], [58.0], [58....
1	151 150 147 155 148 133 111 140 170 174 182 15...	0	1	[0, 1]	[[[151.0], [150.0], [147.0], [155.0], [148.0],...
2	231 212 156 164 174 138 161 173 182 200 106 38...	0	1	[0, 1]	[[[231.0], [212.0], [156.0], [164.0], [174.0],...
3	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	0	1	[0, 1]	[[[24.0], [32.0], [36.0], [30.0], [32.0], [23....
4	4 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	0	1	[0, 1]	[[[4.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0...

In [12]:
```
X = (np.stack (df['pixels3'])) / 255.0

y = np.stack (df.happy_other)
```

In [13]:
```
X_train, X_testval, y_train, y_testval = train_test_split(X,y,test_size=0.2)

X_test, X_val, y_test, y_val = train_test_split(X_testval,y_testval,test_size=0.5)

print('X.shape:', X.shape,'\n''y.shape:', y.shape,'\n')
print('X_train.shape:', X_train.shape,'\n''y_train.shape:', y_train.shape,'\n')
print('X_test.shape:', X_test.shape,'\n''y_test.shape:', y_test.shape,'\n')
print('X_val.shape:', X_val.shape,'\n''y_val.shape:', y_val.shape,'\n')
```

```
X.shape: (35887, 48, 48, 1)
y.shape: (35887, 2)

X_train.shape: (28709, 48, 48, 1)
y_train.shape: (28709, 2)

X_test.shape: (3589, 48, 48, 1)
y_test.shape: (3589, 2)

X_val.shape: (3589, 48, 48, 1)
y_val.shape: (3589, 2)
```

TRAINING our MODEL

In [17]:
```
def base_model():

    model = Sequential()
    input_shape = (48, 48, 1)

    #1st convolution layer
```

```
    model.add(Conv2D(
        filters = 64,
        kernel_size = (5, 5),
        activation ='relu',
        padding ='same'))
    model.add(Conv2D(
        filters = 64,
        kernel_size = (5, 5),
        activation ='relu',
        padding ='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(
        pool_size = (2, 2)))
    model.add(Dropout(0.5))

    #2nd convolution layer
    model.add(Conv2D(
        filters = 128,
        kernel_size = (5, 5),
        activation ='relu',
        padding ='same'))
    model.add(Conv2D(
        filters = 128,
        kernel_size = (5, 5),
        activation ='relu',
        padding ='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(
        pool_size = (2, 2)))
    model.add(Dropout(0.5))

    #3rd convolution layer
    model.add(Conv2D(
        filters = 256,
        kernel_size = (3, 3),
        activation ='relu',
        padding ='same'))
    model.add(Conv2D(
        filters = 256,
        kernel_size = (3, 3),
        activation ='relu',
        padding ='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(
        pool_size = (2, 2)))
    model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(128))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.2))
    model.add(Dense(2))
    model.add(Activation('softmax'))
    #model.add(Dense(1, activation='sigmoid')) (should be used for binary class but giv

    opt = tf.keras.optimizers.Adam(
        learning_rate = 0.001,
        beta_1 = 0.9,
        beta_2 = 0.999,
        epsilon = 1e-07,
        amsgrad = False,
        name ='Adam')

    model.compile(
        loss ='categorical_crossentropy',
        metrics = ['accuracy'],
        optimizer = opt)

    return model
```

Save

In [20]:
```
model_red = base_model()

history = model_red.fit(
    X_train, y_train,
    validation_data = (X_val, y_val),
    #epochs = 15,
    epochs = 3,
    verbose = 1,
    batch_size = 50)

model_red.summary()

model_red.save('../models/model_v4red.h5')

model_json = model_red.to_json()
name_1 = 'model_v4red_weights'
model_red.save_weights(name_1)

with open(name_1+'.json', "w") as json_file:
    json.dump(model_json, json_file)
```

Save

```
Epoch 1/3
575/575 [==============================] - 1835s 3s/step - loss: 0.5317 - accuracy: 0.75
22 - val_loss: 0.5982 - val_accuracy: 0.7623
Epoch 2/3
575/575 [==============================] - 2120s 4s/step - loss: 0.3719 - accuracy: 0.84
14 - val_loss: 0.4307 - val_accuracy: 0.8197
Epoch 3/3
575/575 [==============================] - 1834s 3s/step - loss: 0.3190 - accuracy: 0.86
94 - val_loss: 0.4745 - val_accuracy: 0.8047
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 48, 48, 64)        1664
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 48, 48, 64)        102464
_________________________________________________________________
batch_normalization_12 (Batc (None, 48, 48, 64)        256
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 24, 24, 64)        0
_________________________________________________________________
dropout_12 (Dropout)         (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 24, 24, 128)       204928
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 24, 24, 128)       409728
_________________________________________________________________
batch_normalization_13 (Batc (None, 24, 24, 128)       512
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 12, 12, 128)       0
_________________________________________________________________
dropout_13 (Dropout)         (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 12, 12, 256)       295168
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 12, 12, 256)       590080
_________________________________________________________________
batch_normalization_14 (Batc (None, 12, 12, 256)       1024
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 6, 6, 256)         0
_________________________________________________________________
dropout_14 (Dropout)         (None, 6, 6, 256)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               1179776
_________________________________________________________________
batch_normalization_15 (Batc (None, 128)               512
_________________________________________________________________
activation_6 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0
```

```
dense_7 (Dense)              (None, 2)                 258
_________________________________________________________________
activation_7 (Activation)    (None, 2)                 0
=================================================================
Total params: 2,786,370
Trainable params: 2,785,218
Non-trainable params: 1,152
_________________________________________________________________
```

In [22]:
```
scores = model_red.evaluate(X_test, y_test, verbose=2)
print("Accuracy: %.2f%%" % (scores[1]*100))
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

```
113/113 - 46s - loss: 0.4951 - accuracy: 0.7949
Accuracy: 79.49%
Test loss: 0.495086669921875
Test accuracy: 0.7949289679527283
```

In [28]:
```
Path.cwd().parent
```

Out[28]:
```
PosixPath('/Users/cris/Iron/AudienceResearch')
```

In [33]:
```
Path.cwd().parent/'models/model_v4red.h5', 'r'
```

Out[33]:
```
(PosixPath('/Users/cris/Iron/AudienceResearch/models/model_v4red.h5'), 'r')
```

In [34]:
```
model = h5py.File(Path.cwd().parent/'models/model_v4red.h5', 'r')
```

In [37]:
```
Path.cwd()
```

Out[37]:
```
PosixPath('/Users/cris/Iron/AudienceResearch/notebooks')
```

In [36]:
```
import h5py
import cv2

model = h5py.File(Path.cwd().parent/'models/model_v4red.h5', 'r')
face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
n = 0
counter_fotos = 0
```

In [54]:
```
def chorro(uploaded_file):
    counter_faces = 0

    img = Image.open(uploaded_file)
    new = img.save(Path.cwd().parent/'demo'/'a.jpg')

    new_img = Image.open(Path.cwd().parent/'demo'/'a.jpg')

    #input_img1 = cv2.imread(f"demo/{counter_faces}.jpg")
    input_img1 = cv2.imread('demo/f"{counter_faces}".jpg')
    input_img2 = cv2.cvtColor(input_img1, cv2.COLOR_BGR2GRAY)
    input_img3 = input_img2.copy()

    faceClass = cv2.CascadeClassifier("src/haarcascade_frontalface_default.xml")
    faces = faceClass.detectMultiScale(input_img2,scaleFactor=1.1, minNeighbors=7)
    for (x,y,w,h) in faces:

        counter_faces += 1
```

```
        img_data1 = input_img3 [y:y+h,x:x+w]
        img_data2 = cv2.resize (img_data1, (48, 48))
        img_data3 = np.stack(img_data2)
        img_data4 = img_data2 / 255.0
        img_data5 = np.expand_dims(
            img_data4, axis=0).reshape(
                np.expand_dims(
                    img_data4, axis=0).shape[0], 48, 48, 1)

        cv2.imwrite(f"demo/_{counter_faces}.jpg", img_data2)

        img_datashow = img_data3*255
        img_show = Image.fromarray(img_datashow)
        img = Image.open("images_support/cover1.jpeg")
        img_show.save(f"demo/_{counter_faces}_a.jpg")


        with open(Path.cwd()/'model_v4red_weights.json','r') as f:
            model_json = json.load(f)
            model = model_from_json(model_json)
            model.load_weights('model_v4red_weights.h5.json')
            #model = h5py.File('models/model_v3.hdf5', 'r')
            EM = model.predict(img_data5)[0]
            model_red = load_model('model_v4red.h5')

            counter_faces = 0

            happy = EM[0]
            unhappy = EM[1]
            plt.imshow(Image.fromarray(EM.squeeze()*255))
            st.write("The prediction is… happy:{0:.5f} other:{1:.5f}".format(EM[0],EM[1]
    return "TO BE CONTINUED"
```

Save

In [ ]:

Save

In [ ]:

Save