In [1]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import utils
import os
%matplotlib inline

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Input, Dropout,Flatten, Conv2D
from tensorflow.keras.layers import BatchNormalization, Activation, MaxPooli
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.utils import plot_model

from IPython.display import SVG, Image
from livelossplot.tf_keras import PlotLossesCallback
import tensorflow as tf
print("Tensorflow version:", tf.__version__)
```

```
2023-07-19 21:18:20.506461: I tensorflow/core/platform/cpu_feature_guard.cc:
182] This TensorFlow binary is optimized to use available CPU instructions i
n performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
Tensorflow version: 2.13.0
```

In [2]:
```
data_folder = "data/"

expressions = [expression for expression in os.listdir(data_folder) if os.pa

for expression in expressions:
    print(str(len(os.listdir(data_folder + expression))) + " " + expression
```

```
1774 happy images
1247 sad images
958 angry images
```

In [3]:
```
img_size = 48
batch_size = 64

datagen_train = ImageDataGenerator(horizontal_flip=True)

train_generator = datagen_train.flow_from_directory("data/",
                                                    target_size=(img_size,im
                                                    color_mode="grayscale",
                                                    batch_size=batch_size,
                                                    class_mode='categorical'
                                                    shuffle=True)

datagen_validation = ImageDataGenerator(horizontal_flip=True)
validation_generator = datagen_validation.flow_from_directory("data/",
                                                    target_size=(img_size,im
                                                    color_mode="grayscale",
                                                    batch_size=batch_size,
                                                    class_mode='categorical'
                                                    shuffle=False)
```

```
Found 3979 images belonging to 3 classes.
Found 3979 images belonging to 3 classes.
```

In [4]:
```
# Initialising the CNN
model = Sequential()
```

```
# 1 - Convolution
model.add(Conv2D(64,(3,3), padding='same', input_shape=(48, 48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
# 2nd Convolution layer
model.add(Conv2D(128,(5,5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
# 3rd Convolution layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
# 4th Convolution layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
# Flattening
model.add(Flatten())
# Fully connected layer 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
# Fully connected layer 2nd layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(3, activation='softmax'))
opt = Adam(learning_rate=0.0005)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accu
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 48, 48, 64)        640

 batch_normalization (Batch  (None, 48, 48, 64)        256
 Normalization)

 activation (Activation)     (None, 48, 48, 64)        0

 max_pooling2d (MaxPooling2  (None, 24, 24, 64)        0
 D)

 dropout (Dropout)           (None, 24, 24, 64)        0

 conv2d_1 (Conv2D)           (None, 24, 24, 128)       204928

 batch_normalization_1 (Bat  (None, 24, 24, 128)       512
 chNormalization)

 activation_1 (Activation)   (None, 24, 24, 128)       0

 max_pooling2d_1 (MaxPoolin  (None, 12, 12, 128)       0
 g2D)

 dropout_1 (Dropout)         (None, 12, 12, 128)       0

 conv2d_2 (Conv2D)           (None, 12, 12, 512)       590336

 batch_normalization_2 (Bat  (None, 12, 12, 512)       2048
 chNormalization)

 activation_2 (Activation)   (None, 12, 12, 512)       0

 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 512)         0
 g2D)

 dropout_2 (Dropout)         (None, 6, 6, 512)         0

 conv2d_3 (Conv2D)           (None, 6, 6, 512)         2359808

 batch_normalization_3 (Bat  (None, 6, 6, 512)         2048
 chNormalization)

 activation_3 (Activation)   (None, 6, 6, 512)         0

 max_pooling2d_3 (MaxPoolin  (None, 3, 3, 512)         0
 g2D)

 dropout_3 (Dropout)         (None, 3, 3, 512)         0

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 256)               1179904

 batch_normalization_4 (Bat  (None, 256)               1024
 chNormalization)

 activation_4 (Activation)   (None, 256)               0

 dropout_4 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 512)               131584
```

```
 batch_normalization_5 (Bat  (None, 512)               2048
 chNormalization)

 activation_5 (Activation)   (None, 512)               0

 dropout_5 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 3)                 1539

=================================================================
Total params: 4476675 (17.08 MB)
Trainable params: 4472707 (17.06 MB)
Non-trainable params: 3968 (15.50 KB)
_________________________________________________________________
```

In [5]:

```
%%time

epochs = 15
steps_per_epoch = train_generator.n//train_generator.batch_size
validation_steps = validation_generator.n//validation_generator.batch_size

print(validation_steps)
exit

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
                              patience=2, min_lr=0.00001, mode='auto')
checkpoint = ModelCheckpoint("model_weights.h5", monitor='val_accuracy',
                             save_weights_only=True, mode='max', verbose=1)
callbacks = [PlotLossesCallback(), checkpoint, reduce_lr]

history = model.fit(
    x=train_generator,
    steps_per_epoch=steps_per_epoch,
    epochs=epochs,
    validation_data = validation_generator,
    validation_steps = validation_steps
)
```

```
62
Epoch 1/15
62/62 [==============================] - 70s 1s/step - loss: 1.2601 - accura
cy: 0.4082 - val_loss: 2.6816 - val_accuracy: 0.2641
Epoch 2/15
62/62 [==============================] - 66s 1s/step - loss: 1.1420 - accura
cy: 0.4572 - val_loss: 1.2926 - val_accuracy: 0.4824
Epoch 3/15
62/62 [==============================] - 65s 1s/step - loss: 1.0666 - accura
cy: 0.5160 - val_loss: 0.9908 - val_accuracy: 0.5186
Epoch 4/15
62/62 [==============================] - 66s 1s/step - loss: 1.0044 - accura
cy: 0.5354 - val_loss: 0.9520 - val_accuracy: 0.5421
Epoch 5/15
62/62 [==============================] - 66s 1s/step - loss: 0.9579 - accura
cy: 0.5602 - val_loss: 0.8895 - val_accuracy: 0.5965
Epoch 6/15
62/62 [==============================] - 67s 1s/step - loss: 0.8934 - accura
cy: 0.5905 - val_loss: 0.8320 - val_accuracy: 0.6053
Epoch 7/15
62/62 [==============================] - 66s 1s/step - loss: 0.8472 - accura
cy: 0.6059 - val_loss: 0.7957 - val_accuracy: 0.6247
Epoch 8/15
62/62 [==============================] - 67s 1s/step - loss: 0.8243 - accura
cy: 0.6278 - val_loss: 0.7983 - val_accuracy: 0.6132
Epoch 9/15
62/62 [==============================] - 68s 1s/step - loss: 0.7658 - accura
cy: 0.6544 - val_loss: 0.7151 - val_accuracy: 0.6767
Epoch 10/15
62/62 [==============================] - 67s 1s/step - loss: 0.7401 - accura
cy: 0.6610 - val_loss: 0.6845 - val_accuracy: 0.6968
Epoch 11/15
62/62 [==============================] - 68s 1s/step - loss: 0.7145 - accura
cy: 0.6863 - val_loss: 0.6659 - val_accuracy: 0.6983
Epoch 12/15
62/62 [==============================] - 67s 1s/step - loss: 0.6858 - accura
cy: 0.7006 - val_loss: 0.5905 - val_accuracy: 0.7513
Epoch 13/15
62/62 [==============================] - 68s 1s/step - loss: 0.6379 - accura
cy: 0.7180 - val_loss: 0.7618 - val_accuracy: 0.6648
Epoch 14/15
62/62 [==============================] - 68s 1s/step - loss: 0.6353 - accura
cy: 0.7152 - val_loss: 0.6786 - val_accuracy: 0.7157
Epoch 15/15
62/62 [==============================] - 68s 1s/step - loss: 0.5973 - accura
cy: 0.7458 - val_loss: 0.5297 - val_accuracy: 0.7719
CPU times: user 1h 39min 33s, sys: 12min 16s, total: 1h 51min 49s
Wall time: 16min 46s
```

In [6]:
```
model_json = model.to_json()
model.save_weights('model_facial_recon.h5')
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

In [7]:
```
from tensorflow.keras.models import model_from_json
import numpy as np

import tensorflow as tf


class FacialExpressionModel(object):

    EMOTIONS_LIST = ["Angry","Happy", "Sad"]
```

```
    def __init__(self, model_json_file, model_weights_file):
        # load model from JSON file
        with open(model_json_file, "r") as json_file:
            loaded_model_json = json_file.read()
            self.loaded_model = model_from_json(loaded_model_json)

        # load weights into the new model
        self.loaded_model.load_weights(model_weights_file)
        self.loaded_model.make_predict_function()

    def predict_emotion(self, img):
        self.preds = self.loaded_model.predict(img)
        return FacialExpressionModel.EMOTIONS_LIST[np.argmax(self.preds)]
```

In [8]:

```
import cv2
import numpy as np

facec = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
model = FacialExpressionModel("./model.json", "./model_facial_recon.h5")
font = cv2.FONT_HERSHEY_SIMPLEX

class VideoCamera(object):
    def __init__(self):
        self.video = cv2.VideoCapture(0)

    def __del__(self):
        self.video.release()

    # returns camera frames along with bounding boxes and predictions
    def get_frame(self):
        _, fr = self.video.read()
        gray_fr = cv2.cvtColor(fr, cv2.COLOR_BGR2GRAY)
        faces = facec.detectMultiScale(gray_fr, 1.3, 5)

        for (x, y, w, h) in faces:
            fc = gray_fr[y:y+h, x:x+w]

            roi = cv2.resize(fc, (48, 48))
            pred = model.predict_emotion(roi[np.newaxis, :, :, np.newaxis])

            cv2.putText(fr, pred, (x, y), font, 1, (255, 255, 0), 2)
            cv2.rectangle(fr,(x,y),(x+w,y+h),(255,0,0),2)

        return fr
```

In [9]:

```
def gen(camera):
    while True:
        frame = camera.get_frame()
        cv2.imshow('Facial Expression Recognization',frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cv2.destroyAllWindows()
```

In [ ]:

```
gen(VideoCamera())
```