face-emotion-recognition

March 31, 2024

0.1 Face Emotion Recognition

[10]:
```
#importing important libraries
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import os

# Importing Deep Learning Libraries

from keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import␣
 ↪Dense,Input,Dropout,GlobalAveragePooling2D,Flatten,Conv2D,BatchNormalization,Activation,Max
from keras.models import Model,Sequential
from tensorflow.keras.optimizers import Adam,SGD,RMSprop
```

1 Displaying Images

[11]:
```
#taking inputs images for training
picture_size = 48
folder_path = "../input/face-expression-recognition-dataset/images/"
```

[12]:
```
#checking disgust image
expression = 'disgust'

plt.figure(figsize= (12,12))
for i in range(1, 10, 1):
    plt.subplot(3,3,i)
    img = load_img(folder_path+"train/"+expression+"/"+
                   os.listdir(folder_path + "train/" + expression)[i],␣
 ↪target_size=(picture_size, picture_size))
    plt.imshow(img)
plt.show()
```

2 Making Training and Validation Data

[13]:
```
#splitting data into train, test and validation set
batch_size  = 128

datagen_train  = ImageDataGenerator()
datagen_val = ImageDataGenerator()

train_set = datagen_train.flow_from_directory(folder_path+"train",
                                              target_size =␣
 ↪(picture_size,picture_size),
                                              color_mode = "grayscale",
```

```
                                                batch_size=batch_size,
                                                class_mode='categorical',
                                                shuffle=True)


test_set = datagen_val.flow_from_directory(folder_path+"validation",
                                                target_size =␣
 ↪(picture_size,picture_size),
                                                color_mode = "grayscale",
                                                batch_size=batch_size,
                                                class_mode='categorical',
                                                shuffle=False)
```

```
Found 28821 images belonging to 7 classes.
Found 7066 images belonging to 7 classes.
```

3 Model Building

- Model = sequential : A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.
- Activation = relu :The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.
- padding = The padding parameter of the Keras Conv2D class can take one of two values: 'valid' or 'same'. Setting the value to "valid" parameter means that the input volume is not zero-padded and the spatial dimensions are allowed to reduce via the natural application of convolution.
- Maxpooling = Maximum pooling, or max pooling, is a pooling operation that calculates the maximum, or largest, value in each patch of each feature map. The results are down sampled or pooled feature maps that highlight the most present feature in the patch, not the average presence of the feature in the case of average pooling.
- Batch normalization = Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.
- Dropout = Dropout is a technique used to prevent a model from overfitting. Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0 at each update of the training phase.
- Adam = Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.

- SGD = Stochastic Gradient Descent (SGD) addresses both of these issues by following the negative gradient of the objective after seeing only a single or a few training examples. The use of SGD In the neural network setting is motivated by the high cost of running back propagation over the full training set
- RMSprop = RMSprop is a gradient based optimization technique used in training neural networks. … This normalization balances the step size (momentum), decreasing the step for large gradients to avoid exploding, and increasing the step for small gradients to avoid vanishing.

[14]:
```
#building model with 7 classes
from tensorflow.keras.optimizers import Adam,SGD,RMSprop


no_of_classes = 7

model = Sequential()

#1st CNN layer
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 CNN 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 CNN 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 CNN 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))

model.add(Flatten())
```

```
#Fully connected 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(no_of_classes, activation='softmax'))



opt = Adam(lr = 0.0001)
model.compile(optimizer=opt,loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 48, 48, 64)        640
_________________________________________________________________
batch_normalization_6 (Batch (None, 48, 48, 64)        256
_________________________________________________________________
activation_6 (Activation)    (None, 48, 48, 64)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 24, 24, 64)        0
_________________________________________________________________
dropout_6 (Dropout)          (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 24, 24, 128)       204928
_________________________________________________________________
batch_normalization_7 (Batch (None, 24, 24, 128)       512
_________________________________________________________________
activation_7 (Activation)    (None, 24, 24, 128)       0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
dropout_7 (Dropout)          (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 12, 512)       590336
```

```
_________________________________________________________________
batch_normalization_8 (Batch (None, 12, 12, 512)       2048
_________________________________________________________________
activation_8 (Activation)    (None, 12, 12, 512)       0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 6, 6, 512)         0
_________________________________________________________________
dropout_8 (Dropout)          (None, 6, 6, 512)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 6, 6, 512)         2359808
_________________________________________________________________
batch_normalization_9 (Batch (None, 6, 6, 512)         2048
_________________________________________________________________
activation_9 (Activation)    (None, 6, 6, 512)         0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 3, 3, 512)         0
_________________________________________________________________
dropout_9 (Dropout)          (None, 3, 3, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               1179904
_________________________________________________________________
batch_normalization_10 (Batc (None, 256)               1024
_________________________________________________________________
activation_10 (Activation)   (None, 256)               0
_________________________________________________________________
dropout_10 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               131584
_________________________________________________________________
batch_normalization_11 (Batc (None, 512)               2048
_________________________________________________________________
activation_11 (Activation)   (None, 512)               0
_________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 7)                 3591
=================================================================
Total params: 4,478,727
Trainable params: 4,474,759
Non-trainable params: 3,968
_________________________________________________________________
/opt/conda/lib/python3.7/site-packages/keras/optimizer_v2/optimizer_v2.py:356:
UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

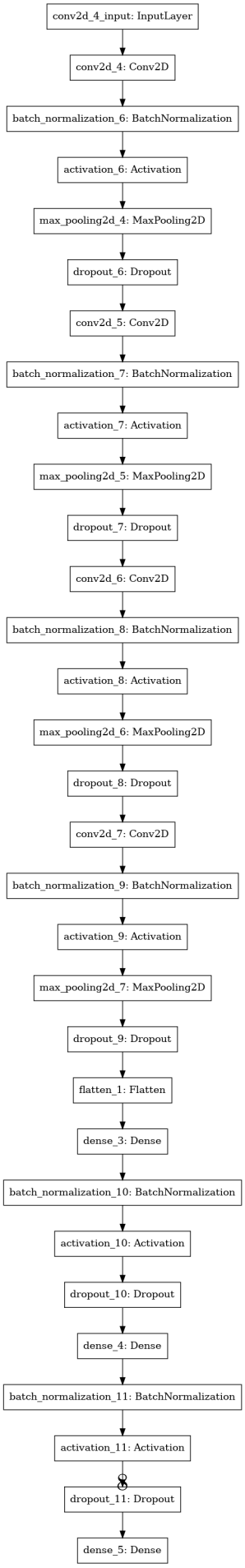
4 Visualize model

The plot_model() function in Keras will create a plot of your network. This function takes a few useful arguments:

- model: (required) The model that you wish to plot.
- to_file: (required) The name of the file to which to save the plot.
- show_shapes: (optional, defaults to False) Whether or not to show the output shapes of each layer.
- show_layer_names: (optional, defaults to True) Whether or not to show the name for each lay

[15]:
```
#visualizing the model
import tensorflow as tf
tf.keras.utils.plot_model(
    model,
    to_file="model.png",
    show_shapes=False,
    show_dtype=False,
    show_layer_names=True,
    rankdir="TB",
    expand_nested=False,
    dpi=96,
    layer_range=None,
)
```

[15]:


conv2d_4_input: InputLayer
conv2d_4: Conv2D
batch_normalization_6: BatchNormalization
activation_6: Activation
max_pooling2d_4: MaxPooling2D
dropout_6: Dropout
conv2d_5: Conv2D
batch_normalization_7: BatchNormalization
activation_7: Activation
max_pooling2d_5: MaxPooling2D
dropout_7: Dropout
conv2d_6: Conv2D
batch_normalization_8: BatchNormalization
activation_8: Activation
max_pooling2d_6: MaxPooling2D
dropout_8: Dropout
conv2d_7: Conv2D
batch_normalization_9: BatchNormalization
activation_9: Activation
max_pooling2d_7: MaxPooling2D
dropout_9: Dropout
flatten_1: Flatten
dense_3: Dense
batch_normalization_10: BatchNormalization
activation_10: Activation
dropout_10: Dropout
dense_4: Dense
batch_normalization_11: BatchNormalization
activation_11: Activation
dropout_11: Dropout
dense_5: Dense

5 Fitting the Model with Training and Validation Data

[16]:
```
#importing tensorflow library and package
from tensorflow.keras.optimizers import RMSprop,SGD,Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,␣
 ↪ReduceLROnPlateau

checkpoint = ModelCheckpoint("./model.h5", monitor='val_acc', verbose=1,␣
 ↪save_best_only=True, mode='max')
#Stopping training when a monitored metric has stopped improving.
early_stopping = EarlyStopping(monitor='val_loss',
                           min_delta=0,
                           patience=3,
                           verbose=1,
                           restore_best_weights=True
                           )

reduce_learningrate = ReduceLROnPlateau(monitor='val_loss',
                                factor=0.2,
                                patience=3,
                                verbose=1,
                                min_delta=0.0001)

callbacks_list = [early_stopping,checkpoint,reduce_learningrate]

epochs = 50

model.compile(loss='categorical_crossentropy',
              optimizer = Adam(lr=0.001),
              metrics=['accuracy'])
```

```
/opt/conda/lib/python3.7/site-packages/keras/optimizer_v2/optimizer_v2.py:356:
UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

[17]:
```
#fitting model with 48 epoch
history = model.fit_generator(generator=train_set,
                              steps_per_epoch=train_set.n//train_set.
 ↪batch_size,
                              epochs=epochs,
                              validation_data = test_set,
                              validation_steps = test_set.n//test_set.
 ↪batch_size,
```

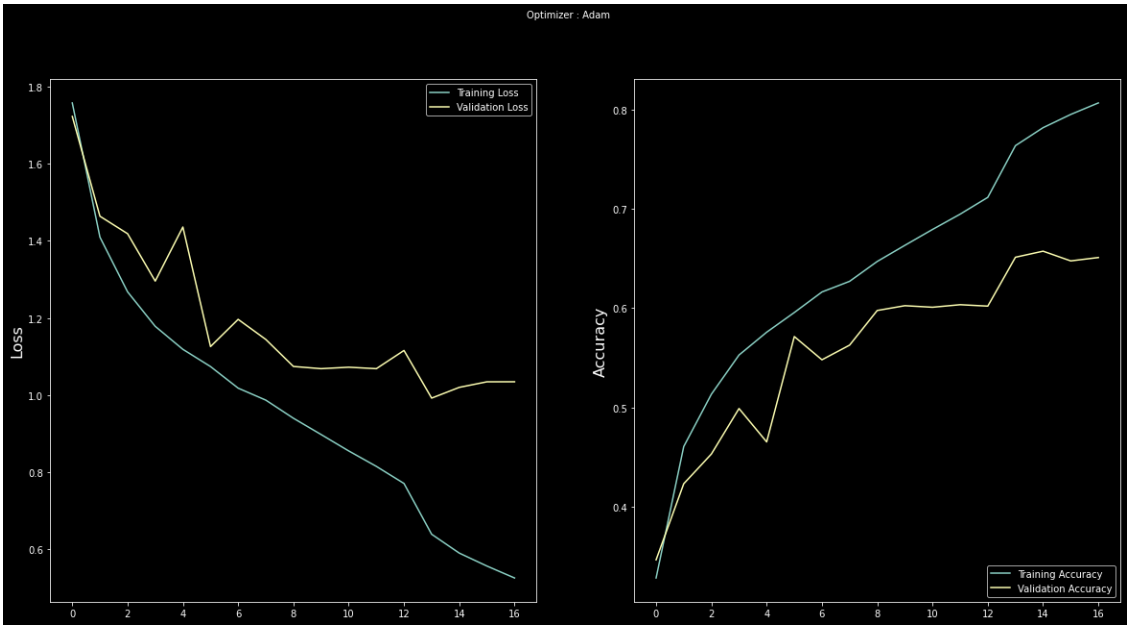
```
                                callbacks=callbacks_list
                                )
```

```
/opt/conda/lib/python3.7/site-packages/keras/engine/training.py:1972:
UserWarning: `Model.fit_generator` is deprecated and will be removed in a future
version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/50
225/225 [==============================] - 32s 136ms/step - loss: 1.7579 -
accuracy: 0.3280 - val_loss: 1.7229 - val_accuracy: 0.3463
Epoch 2/50
225/225 [==============================] - 30s 134ms/step - loss: 1.4099 -
accuracy: 0.4605 - val_loss: 1.4641 - val_accuracy: 0.4230
Epoch 3/50
225/225 [==============================] - 30s 131ms/step - loss: 1.2680 -
accuracy: 0.5133 - val_loss: 1.4188 - val_accuracy: 0.4530
Epoch 4/50
225/225 [==============================] - 35s 154ms/step - loss: 1.1786 -
accuracy: 0.5528 - val_loss: 1.2958 - val_accuracy: 0.4989
Epoch 5/50
225/225 [==============================] - 30s 134ms/step - loss: 1.1190 -
accuracy: 0.5758 - val_loss: 1.4361 - val_accuracy: 0.4652
Epoch 6/50
225/225 [==============================] - 30s 131ms/step - loss: 1.0743 -
accuracy: 0.5957 - val_loss: 1.1260 - val_accuracy: 0.5714
Epoch 7/50
225/225 [==============================] - 30s 135ms/step - loss: 1.0183 -
accuracy: 0.6164 - val_loss: 1.1967 - val_accuracy: 0.5480
Epoch 8/50
225/225 [==============================] - 30s 133ms/step - loss: 0.9873 -
accuracy: 0.6271 - val_loss: 1.1446 - val_accuracy: 0.5628
Epoch 9/50
225/225 [==============================] - 30s 132ms/step - loss: 0.9402 -
accuracy: 0.6472 - val_loss: 1.0745 - val_accuracy: 0.5977
Epoch 10/50
225/225 [==============================] - 30s 133ms/step - loss: 0.8981 -
accuracy: 0.6634 - val_loss: 1.0688 - val_accuracy: 0.6026
Epoch 11/50
225/225 [==============================] - 30s 132ms/step - loss: 0.8555 -
accuracy: 0.6794 - val_loss: 1.0726 - val_accuracy: 0.6010
Epoch 12/50
225/225 [==============================] - 30s 133ms/step - loss: 0.8154 -
accuracy: 0.6949 - val_loss: 1.0688 - val_accuracy: 0.6036
Epoch 13/50
225/225 [==============================] - 29s 130ms/step - loss: 0.7711 -
accuracy: 0.7118 - val_loss: 1.1160 - val_accuracy: 0.6021
```

```
Epoch 00013: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 14/50
225/225 [==============================] - 30s 135ms/step - loss: 0.6395 -
accuracy: 0.7639 - val_loss: 0.9925 - val_accuracy: 0.6514
Epoch 15/50
225/225 [==============================] - 30s 133ms/step - loss: 0.5905 -
accuracy: 0.7821 - val_loss: 1.0203 - val_accuracy: 0.6575
Epoch 16/50
225/225 [==============================] - 29s 130ms/step - loss: 0.5570 -
accuracy: 0.7954 - val_loss: 1.0347 - val_accuracy: 0.6477
Epoch 17/50
225/225 [==============================] - 30s 132ms/step - loss: 0.5260 -
accuracy: 0.8070 - val_loss: 1.0348 - val_accuracy: 0.6511
Restoring model weights from the end of the best epoch.

Epoch 00017: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
Epoch 00017: early stopping
```

6 Plotting Accuracy & Loss

[18]:
```
#plotting graph to check accuracy and loss
plt.style.use('dark_background')

plt.figure(figsize=(20,10))
plt.subplot(1, 2, 1)
plt.suptitle('Optimizer : Adam', fontsize=10)
plt.ylabel('Loss', fontsize=16)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Accuracy', fontsize=16)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='lower right')
plt.show()
```


Optimizer : Adam
Training Loss
Validation Loss
Loss
Accuracy
Training Accuracy
Validation Accuracy

[ ]: