TOPIC : EMOTION AI

AIM :

The main aim of this project / case study is to generate a model that can be used to detect the emotion of a person using his/her facial points from a gray scale image.

AUTHOR : ADITYA KHARE

This project/case study is divided into several smaller tasks which are further divied into various parts which casn help to indentify the progress of the project more easily and will be convinent to track.

Task 01 : Understanding the problem statement and business case

Project Overview :

1. The aim of this project is to classify people's emotions based on their face images.
2. In this case study, I am going to assume that I am working as an AI/ML consultant.
3. I have been hired by a startup to build, train and deploy a system that automatically monitors people emotions and expressions. For this a team shas collected more than 20,000 facial images, with their associated facial expression labels and around 2,000 images with their facial key-point annotations.

Part 01 : Key Facial Points Detection

1. In this we will create a deep learning model based on Convolutional Neural Network and Residual Blocks to predict the facial key-points.
2. the dataset consists of x and y coordinates of facial key-points.
3. Input images are 96 x 96 pixels.
4. Images consists of only one color channel (gray-scale images).

Task 02 : Importing Libraries and Datasets

```
1 # Mounting the drive
2 from google.colab import drive
3 drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
%cd /content/drive/My Drive/AI Projects/Emotion-AI/
```

```
/content/drive/My Drive/AI Projects/Emotion-AI
```

```
# Importing the necessary packages

import pandas as pd
import numpy as np
import os
import PIL
import seaborn as sns
import pickle
from PIL import *
import cv2
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoin
from IPython.display import display
from tensorflow.python.keras import *
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, optimizers
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import *
from tensorflow.keras import backend as K
from keras import optimizers
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from google.colab.patches import cv2_imshow
```

```
# loading facial key points data
keyfacial_df = pd.read_csv('/content/drive/My Drive/AI Projects/Emotion-AI/Emotion AI D
```

```
keyfacial_df
```

	left_eye_center_x	left_eye_center_y	right_eye_center_x	right_eye_center_y
0	66.033564	39.002274	30.227008	36.421678
1	64.332936	34.970077	29.949277	33.448715
2	65.057053	34.909642	30.903789	34.909642
3	65.225739	37.261774	32.023096	37.261774
4	66.725301	39.621261	32.244810	38.042032

...	...	...	...	...
2135	67.180378	35.816373	33.239956	34.921932
2136	65.724490	36.301020	25.377551	37.311224
2137	68.430866	38.651975	28.895857	37.617027

```
1 # obtaining relavant information related to the the dataframe
2 keyfacial_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2140 entries, 0 to 2139
Data columns (total 31 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   left_eye_center_x         2140 non-null   float64
 1   left_eye_center_y         2140 non-null   float64
 2   right_eye_center_x        2140 non-null   float64
 3   right_eye_center_y        2140 non-null   float64
 4   left_eye_inner_corner_x   2140 non-null   float64
 5   left_eye_inner_corner_y   2140 non-null   float64
 6   left_eye_outer_corner_x   2140 non-null   float64
 7   left_eye_outer_corner_y   2140 non-null   float64
 8   right_eye_inner_corner_x  2140 non-null   float64
 9   right_eye_inner_corner_y  2140 non-null   float64
 10  right_eye_outer_corner_x  2140 non-null   float64
 11  right_eye_outer_corner_y  2140 non-null   float64
 12  left_eyebrow_inner_end_x  2140 non-null   float64
```

```
 13  left_eyebrow_inner_end_y   2140 non-null   float64
 14  left_eyebrow_outer_end_x   2140 non-null   float64
 15  left_eyebrow_outer_end_y   2140 non-null   float64
 16  right_eyebrow_inner_end_x  2140 non-null   float64
 17  right_eyebrow_inner_end_y  2140 non-null   float64
 18  right_eyebrow_outer_end_x  2140 non-null   float64
 19  right_eyebrow_outer_end_y  2140 non-null   float64
 20  nose_tip_x                 2140 non-null   float64
 21  nose_tip_y                 2140 non-null   float64
 22  mouth_left_corner_x        2140 non-null   float64
 23  mouth_left_corner_y        2140 non-null   float64
 24  mouth_right_corner_x       2140 non-null   float64
 25  mouth_right_corner_y       2140 non-null   float64
 26  mouth_center_top_lip_x     2140 non-null   float64
 27  mouth_center_top_lip_y     2140 non-null   float64
 28  mouth_center_bottom_lip_x  2140 non-null   float64
 29  mouth_center_bottom_lip_y  2140 non-null   float64
 30  Image                      2140 non-null   object
dtypes: float64(30), object(1)
memory usage: 518.4+ KB
```

```
1 # check if null values exist in the dataframe
2 keyfacial_df.isnull().sum()
```

```
left_eye_center_x            0
left_eye_center_y            0
right_eye_center_x           0
right_eye_center_y           0
left_eye_inner_corner_x      0
left_eye_inner_corner_y      0
left_eye_outer_corner_x      0
left_eye_outer_corner_y      0
right_eye_inner_corner_x     0
right_eye_inner_corner_y     0
right_eye_outer_corner_x     0
right_eye_outer_corner_y     0
left_eyebrow_inner_end_x     0
left_eyebrow_inner_end_y     0
left_eyebrow_outer_end_x     0
left_eyebrow_outer_end_y     0
right_eyebrow_inner_end_x    0
right_eyebrow_inner_end_y    0
right_eyebrow_outer_end_x    0
right_eyebrow_outer_end_y    0
nose_tip_x                   0
nose_tip_y                   0
mouth_left_corner_x          0
mouth_left_corner_y          0
mouth_right_corner_x         0
mouth_right_corner_y         0
mouth_center_top_lip_x       0
mouth_center_top_lip_y       0
mouth_center_bottom_lip_x    0
mouth_center_bottom_lip_y    0
Image                        0
dtype: int64
```

```
1 keyfacial_df['Image'].shape
```

```
(2140,)
```

```
# Since values for the image are given as space separated string, separate the values u
# Then convert this into numpy array using np.fromstring and convert the obtained 1D ar
keyfacial_df['Image'] = keyfacial_df['Image'].apply(lambda x: np.fromstring(x, dtype =
```

```
# Obtain the Shape of the image
keyfacial_df['Image'][0].shape
```

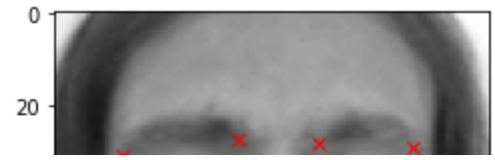
```
(96, 96)
```

```
keyfacial_df.describe()
```

	left_eye_center_x	left_eye_center_y	right_eye_center_x	right_eye_center_y
count	2140.000000	2140.000000	2140.000000	2140.000000
mean	66.221549	36.842274	29.640269	37.063815
std	2.087683	2.294027	2.051575	2.234334
min	47.835757	23.832996	18.922611	24.773072
25%	65.046300	35.468842	28.472224	35.818377
50%	66.129065	36.913319	29.655440	37.048085
75%	67.332093	38.286438	30.858673	38.333884
max	78.013082	46.132421	42.495172	45.980981

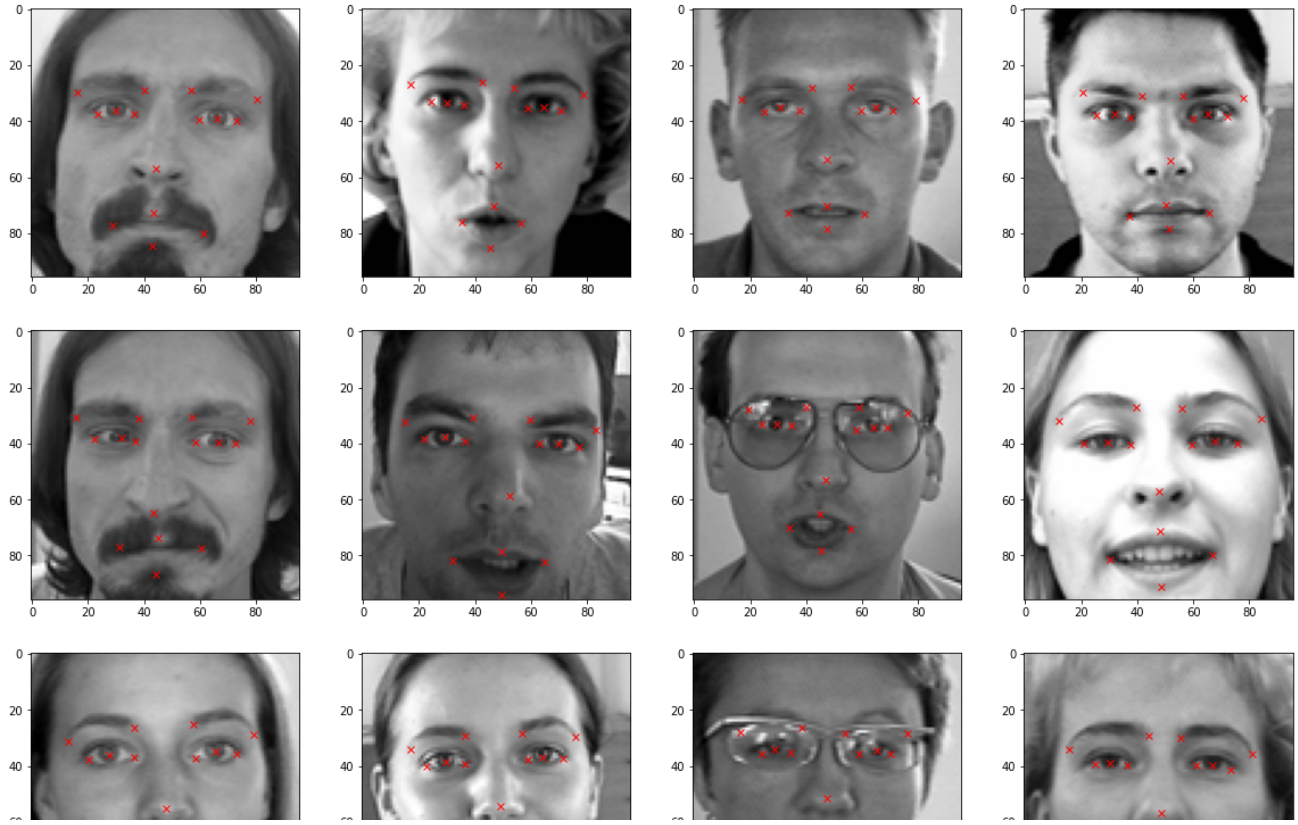
TASK 03: PERFORM IMAGE VISUALIZATION

```
# Plot a random image from the dataset along with facial keypoints.
# Image data is obtained from df['Image'] and plotted using plt.imshow
# 15 x and y coordinates for the corresponding image
# since x-coordinates are in even columns like 0,2,4,.. and y-coordinates are in odd co
# we access their value using .loc command, which get the values for coordinates of the

i = np.random.randint(1, len(keyfacial_df))
plt.imshow(keyfacial_df['Image'][i], cmap = 'gray')
for j in range(1, 31, 2):
        plt.plot(keyfacial_df.loc[i][j-1], keyfacial_df.loc[i][j], 'rx')
```



Double-click (or enter) to edit

```
# Let's view more images in a grid format
fig = plt.figure(figsize=(20, 20))

for i in range(16):
    ax = fig.add_subplot(4, 4, i + 1)
    image = plt.imshow(keyfacial_df['Image'][i],cmap = 'gray')
    for j in range(1,31,2):
        plt.plot(keyfacial_df.loc[i][j-1], keyfacial_df.loc[i][j], 'rx')
```



MINI CHALLENGE 02:

Perform a sanity check on the data by randomly visualizing 64 new images along with their cooresponding key points

```
import random
# Let's view more images in a grid format
fig = plt.figure(figsize=(20, 20))

for i in range(64):
    k = random.randint(1, len(keyfacial_df))
    ax = fig.add_subplot(8, 8, i + 1)
    image = plt.imshow(keyfacial_df['Image'][k],cmap = 'gray')
    for j in range(1,31,2):
        plt.plot(keyfacial_df.loc[k][j-1], keyfacial_df.loc[k][j], 'rx')
```

TASK 04: PERFORM IMAGE AUGMENTATION

```
1 # Create a new copy of the dataframe
2 import copy
3 keyfacial_df_copy = copy.copy(keyfacial_df)
```

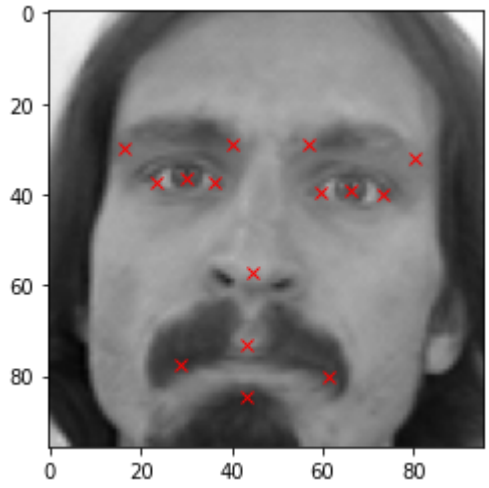
```
1 # Obtain the columns in the dataframe
```

```
2
3 columns = keyfacial_df_copy.columns[:-1]
4 columns
```

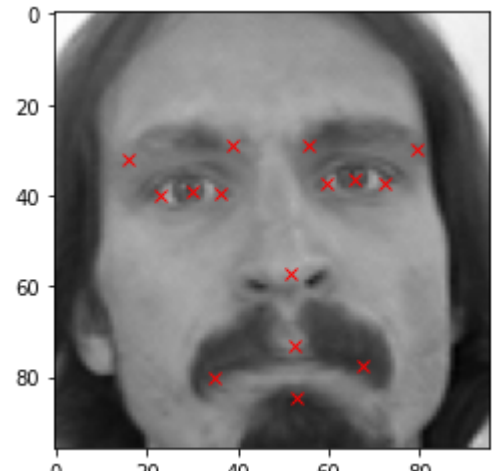
```
Index(['left_eye_center_x', 'left_eye_center_y', 'right_eye_center_x',
       'right_eye_center_y', 'left_eye_inner_corner_x',
       'left_eye_inner_corner_y', 'left_eye_outer_corner_x',
       'left_eye_outer_corner_y', 'right_eye_inner_corner_x',
       'right_eye_inner_corner_y', 'right_eye_outer_corner_x',
       'right_eye_outer_corner_y', 'left_eyebrow_inner_end_x',
       'left_eyebrow_inner_end_y', 'left_eyebrow_outer_end_x',
       'left_eyebrow_outer_end_y', 'right_eyebrow_inner_end_x',
       'right_eyebrow_inner_end_y', 'right_eyebrow_outer_end_x',
       'right_eyebrow_outer_end_y', 'nose_tip_x', 'nose_tip_y',
       'mouth_left_corner_x', 'mouth_left_corner_y', 'mouth_right_corner_x',
       'mouth_right_corner_y', 'mouth_center_top_lip_x',
       'mouth_center_top_lip_y', 'mouth_center_bottom_lip_x',
       'mouth_center_bottom_lip_y'],
      dtype='object')
```

```
1 # Horizontal Flip - flip the images along y axis
2 keyfacial_df_copy['Image'] = keyfacial_df_copy['Image'].apply(lambda x: np.flip(x, axis
3
4 # since we are flipping horizontally, y coordinate values would be the same
5 # Only x coordiante values would change, all we have to do is to subtract our initial x
6 for i in range(len(columns)):
7   if i%2 == 0:
8     keyfacial_df_copy[columns[i]] = keyfacial_df_copy[columns[i]].apply(lambda x: 96. -
```

```
1 # Show the Original image
2 plt.imshow(keyfacial_df['Image'][0], cmap = 'gray')
3 for j in range(1, 31, 2):
4         plt.plot(keyfacial_df.loc[0][j-1], keyfacial_df.loc[0][j], 'rx')
```




```
1 # Show the Horizontally flipped image
2 plt.imshow(keyfacial_df_copy['Image'][0],cmap='gray')
3 for j in range(1, 31, 2):
4         plt.plot(keyfacial_df_copy.loc[0][j-1], keyfacial_df_copy.loc[0][j], 'rx')
```




```
# Concatenate the original dataframe with the augmented dataframe
augmented_df = np.concatenate((keyfacial_df, keyfacial_df_copy))
```

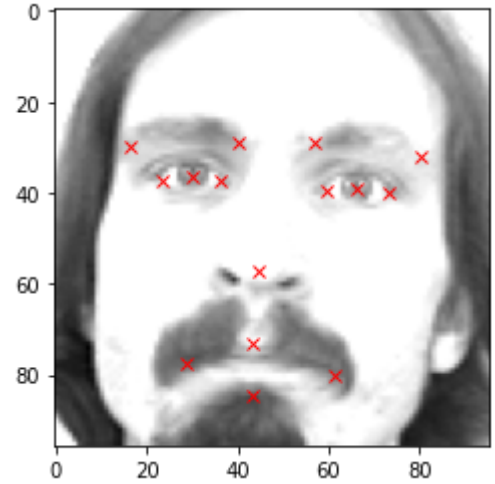
```
augmented_df.shape
```

```
(4280, 31)
```

```
# Randomingly increasing the brightness of the images
# We multiply pixel values by random values between 1.5 and 2 to increase the brightnes
# we clip the value between 0 and 255

import random

keyfacial_df_copy = copy.copy(keyfacial_df)
keyfacial_df_copy['Image'] = keyfacial_df_copy['Image'].apply(lambda x:np.clip(random.u
augmented_df = np.concatenate((augmented_df, keyfacial_df_copy))
augmented_df.shape
```

```
(6420, 31)
```

```
# Show Image with increased brightness

plt.imshow(keyfacial_df_copy['Image'][0], cmap='gray')
for j in range(1, 31, 2):
        plt.plot(keyfacial_df_copy.loc[0][j-1], keyfacial_df_copy.loc[0][j], 'rx')
```



MINI CHALLENGE 03:

- Augment images by flipping them vertically (Hint: Flip along x-axis and note that if we are flipping along x-axis, x co-ordinates won't change)

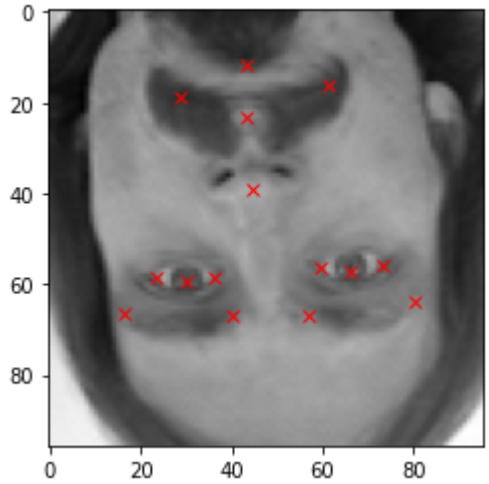
```
keyfacial_df_copy = copy.copy(keyfacial_df)
```

```
keyfacial_df_copy['Image'] = keyfacial_df_copy['Image'].apply(lambda x: np.flip(x, axis

for i in range(len(columns)):
  if i%2 == 1:
    keyfacial_df_copy[columns[i]] = keyfacial_df_copy[columns[i]].apply(lambda x: 96. -
```

MINI CHALLENGE 04:

- Perform a sanity check and visualize sample images

```
plt.imshow(keyfacial_df_copy['Image'][0], cmap='gray')
for j in range(1, 31, 2):
        plt.plot(keyfacial_df_copy.loc[0][j-1], keyfacial_df_copy.loc[0][j], 'rx')
```



TASK 05: PERFORM DATA NORMALIZATION AND TRAINING DATA PREPARATION

```
# Obtain the value of images which is present in the 31st column (since index start fro
img = augmented_df[:,30]

# Normalize the images
img = img/255.

# Create an empty array of shape (x, 96, 96, 1) to feed the model
X = np.empty((len(img), 96, 96, 1))
```

```
9
10 # Iterate through the img list and add image values to the empty array after expanding
11 for i in range(len(img)):
12   X[i,] = np.expand_dims(img[i], axis = 2)
13
14 # Convert the array type to float32
15 X = np.asarray(X).astype(np.float32)
16 X.shape
```

```
(6420, 96, 96, 1)
```

```
1 # Obtain the value of x & y coordinates which are to used as target.
2 y = augmented_df[:,:30]
3 y = np.asarray(y).astype(np.float32)
4 y.shape
```

```
(6420, 30)
```

```
1 # Split the data into train and test data
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

MINI CHALLENGE 05:

Try a different value for 'test_size' and verify that the split was successful

```
1 X_train.shape
```

```
(5136, 96, 96, 1)
```

```
1 X_test.shape
```

```
(1284, 96, 96, 1)
```

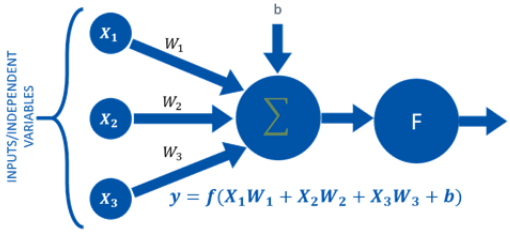
TASK 06: UNDERSTAND THE THEORY AND INTUITION BEHIND NEURAL NETWORKS

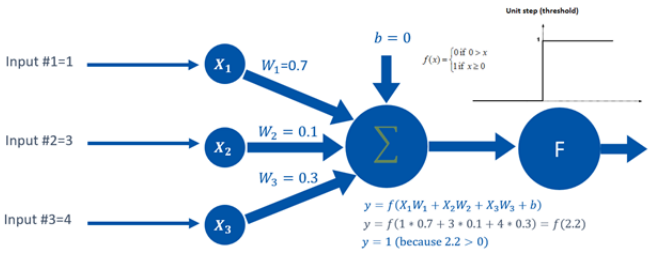
NEURON MATHEMATICAL MODEL

- The brain has over 100 billion neurons communicating through electrical & chemical signals. Neurons communicate with each other and help us see, think, and generate ideas.
- Human brain learns by creating connections among these neurons. ANNs are information processing models inspired by the human brain.
- The neuron collects signals from input channels named dendrites, processes information in its nucleus, and then generates an output in

NEURON MATHEMATICAL MODEL: EXAMPLE

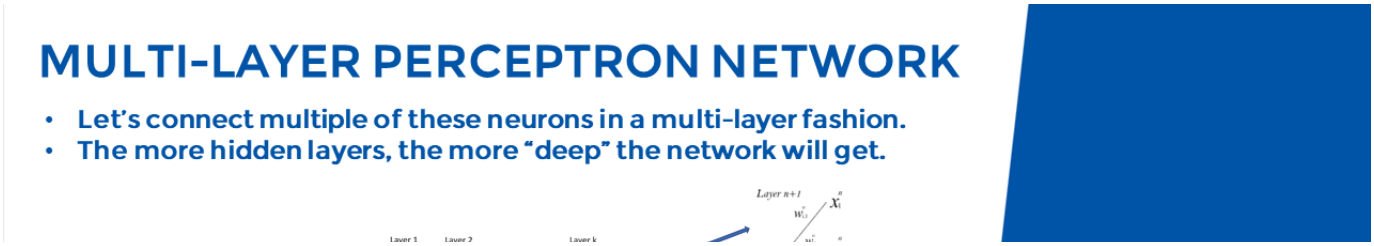
- Bias allows to shift the activation function curve up or down.
- Number of adjustable parameters = 4 (3 weights and 1 bias).
- Activation function "F".





MINI CHALLENGE 06:

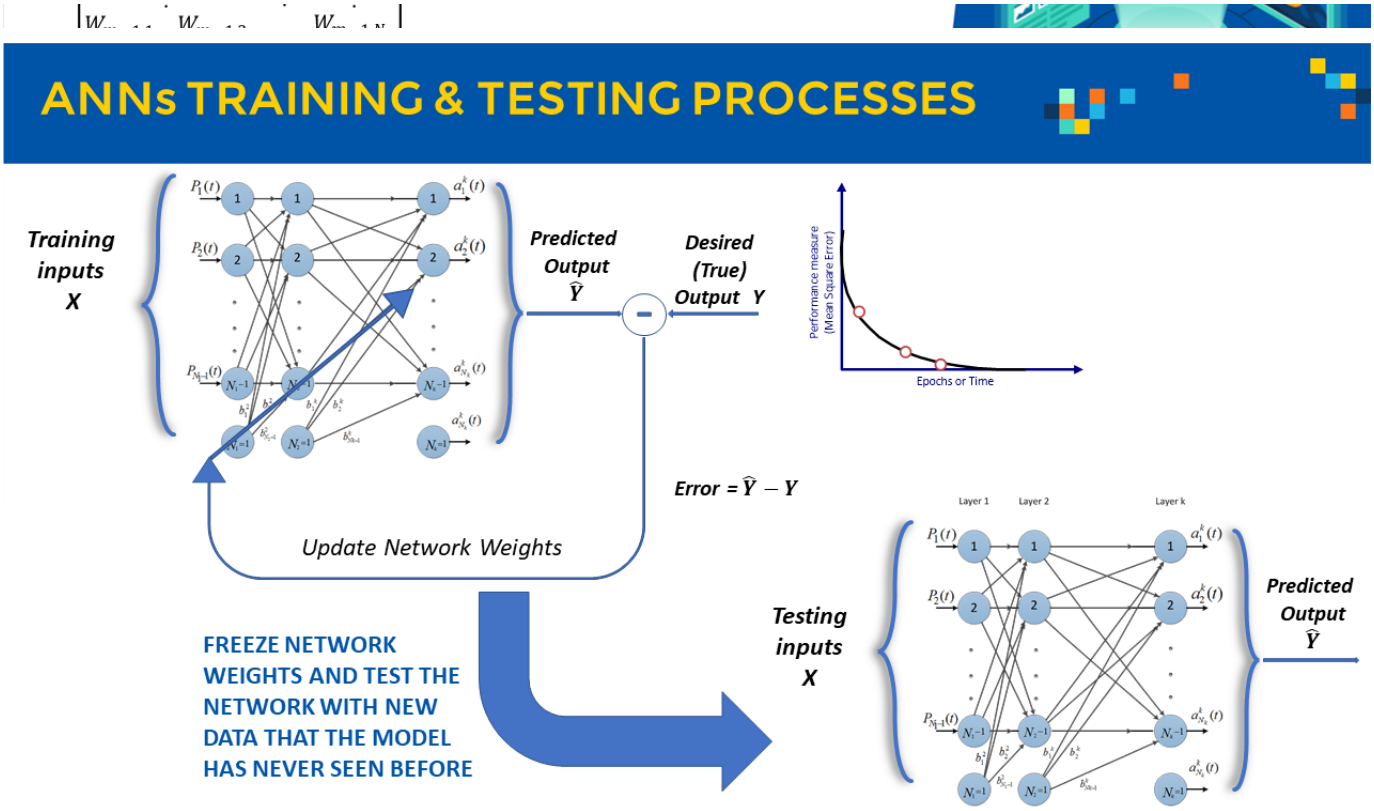
- List at least 3 types of activation functions, which type is preferred in the hidden layers?



MINI CHALLENGE #7:

- List at least 3 different neural networks architectures and their applications.

TASK 7: UNDERSTAND NEURAL NETWORKS TRAINING PROCESS AND GRADIENT DESCENT ALGORITHM



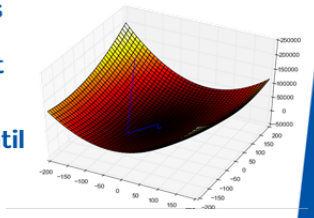
DIVIDE DATA INTO TRAINING AND TESTING

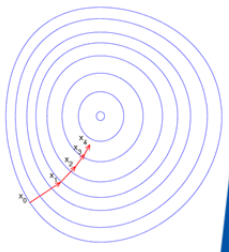
- Data set is generally divided into 80% for training and 20% for testing.
- Sometimes, we might include cross validation dataset as well and then we

TRAINING DATASET 80%

GRADIENT DESCENT

- Gradient descent is an optimization algorithm used to obtain the optimized network weight and bias values
- It works by iteratively trying to minimize the cost function
- It works by calculating the gradient of the cost function and moving in the negative direction until the local/global minimum is achieved
- If the positive of the gradient is taken, local/global maximum is achieved
- The size of the steps taken are called the learning rate
- If learning rate increases, the area covered in the search space will increase so we might reach global minimum faster
- However, we can overshoot the target
- For small learning rates, training will take much longer to reach optimized weight values





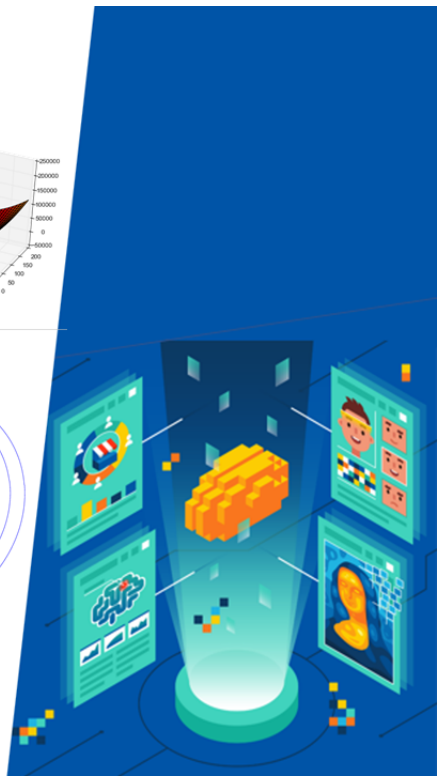


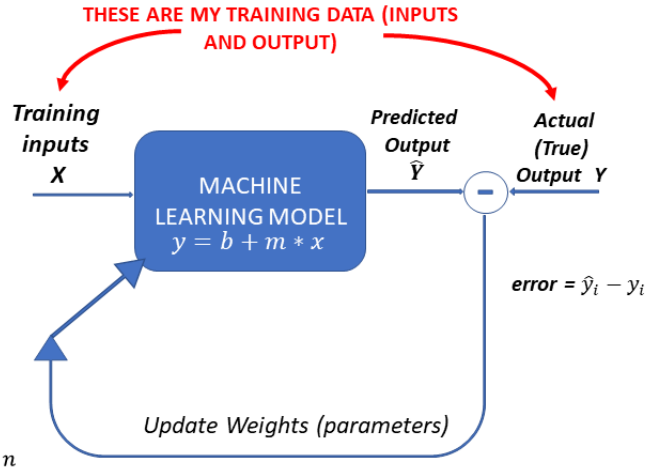
Photo Credit: https://commons.wikimedia.org/wiki/File:Gradient_descent_method.png
Photo Credit: https://commons.wikimedia.org/wiki/File:Gradient_descent.png

GRADIENT DESCENT

- Let's assume that we want to obtain the optimal values for parameters 'm' and 'b'.

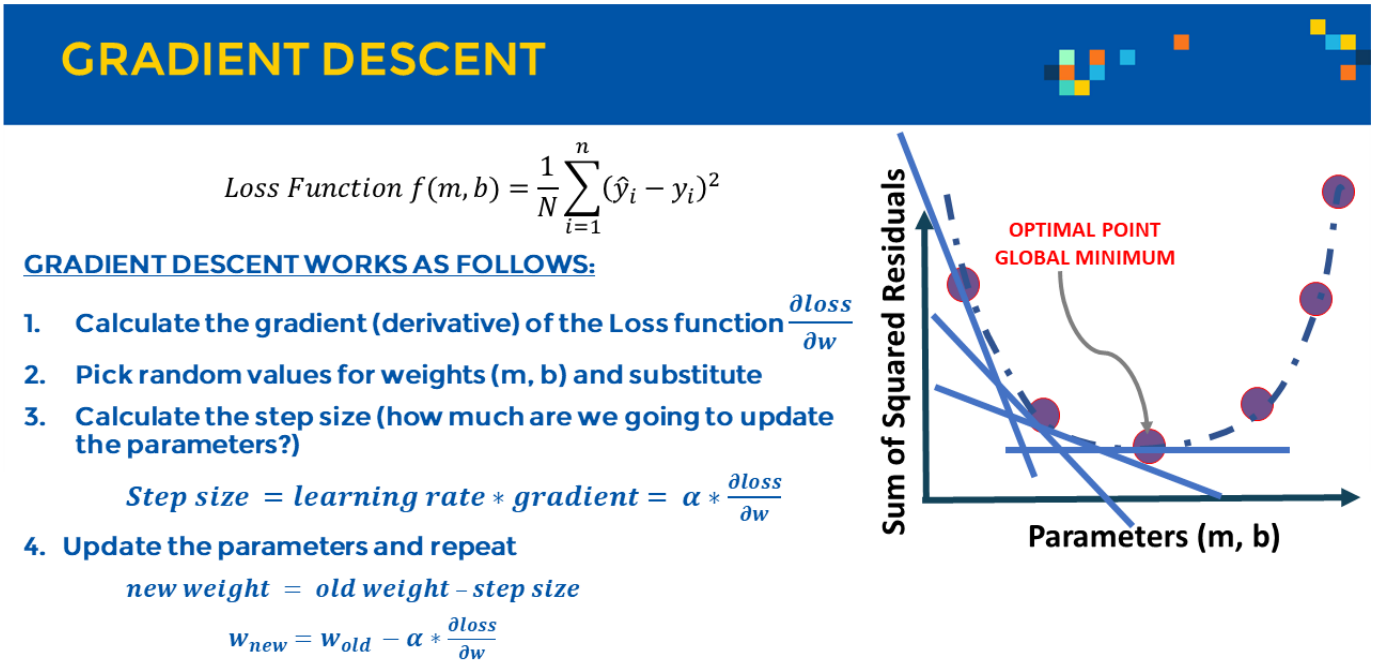
$$y = b + m * x$$

GOAL IS TO FIND BEST PARAMETERS



- We need to first formulate a loss function as follows:

$$Cost\ Function\ f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(error)^2 = \frac{1}{N}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$



MINI CHALLENGE 08:

- What happen when you set the learning rate to the extremes (i.e.: very small value and very large value)? How can you achieve the best of both worlds?

TASK 08: UNDERSTAND THE THEORY AND INTUITION BEHIND CONVOLUTIONAL NEURAL NETWORKS AND RESNETS

CONVOLUTIONAL NEURAL NETWORKS: ENTIRE NETWORK OVERVIEW

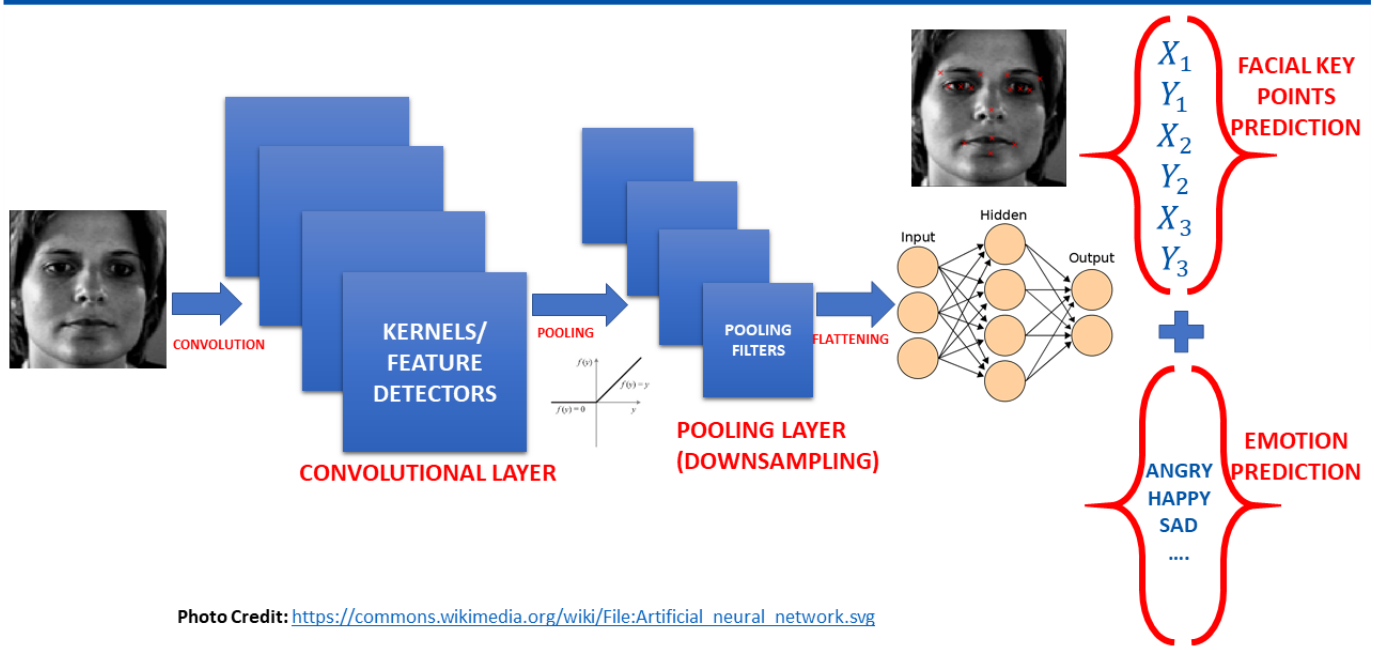
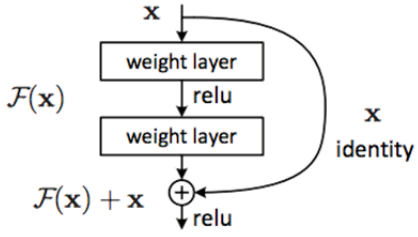
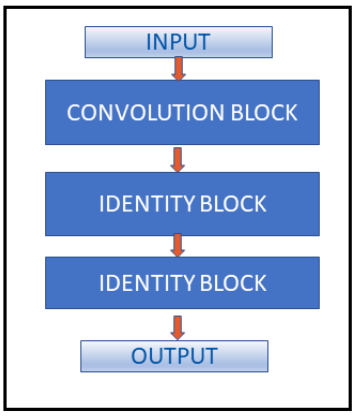


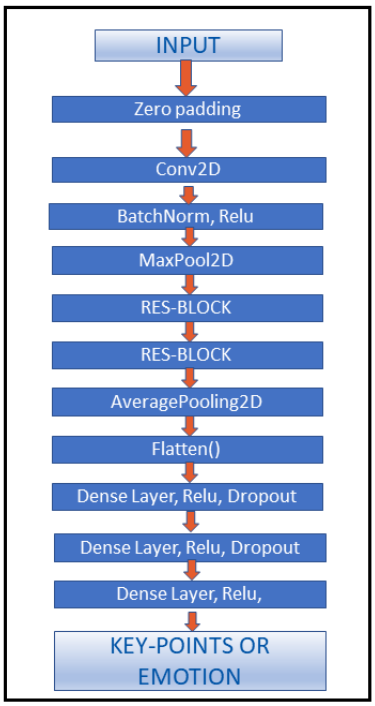
Photo Credit: https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg

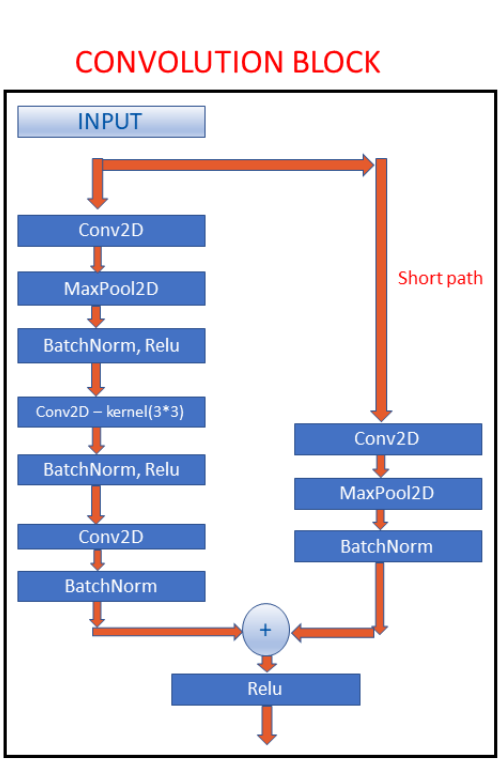
RESNET (RESIDUAL NETWORK)

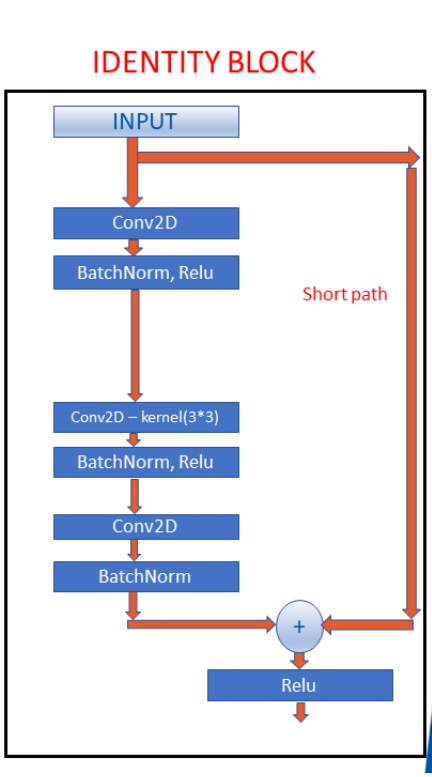
- As CNNs grow deeper, vanishing gradient tend to occur which negatively impact network performance.
- Vanishing gradient problem occurs when the gradient is back-propagated to earlier layers which results in a very small gradient.
- Residual Neural Network includes "skip connection" feature which enables training of 152 layers without vanishing gradient issues.
- Resnet works by adding "identity mappings" on top of CNN.
- ImageNet contains 11 million images and 11,000 categories.
- ImageNet is used to train ResNet deep network.




RES-BLOCK
INPUT
CONVOLUTION BLOCK
IDENTITY BLOCK
IDENTITY BLOCK
OUTPUT


FINAL MODEL
INPUT
Zero padding
Conv2D
BatchNorm, Relu
MaxPool2D
RES-BLOCK
RES-BLOCK
AveragePooling2D
Flatten()
Dense Layer, Relu, Dropout
Dense Layer, Relu, Dropout
Dense Layer, Relu,
KEY-POINTS OR EMOTION


CONVOLUTION BLOCK
INPUT
Conv2D
MaxPool2D
BatchNorm, Relu
Conv2D – kernel(3*3)
BatchNorm, Relu
Conv2D
BatchNorm
Short path
Conv2D
MaxPool2D
BatchNorm
+
Relu


IDENTITY BLOCK
INPUT
Conv2D
BatchNorm, Relu
Short path
Conv2D – kernel(3*3)
BatchNorm, Relu
Conv2D
BatchNorm
+
Relu

MINI CHALLENGE 09:

- What's the performance of ResNets compared to AlexNet in ImageNet dataset?

TASK 09: BUILD DEEP RESIDUAL NEURAL NETWORK KEY FACIAL POINTS DETECTION MODEL

```
def res_block(X, filter, stage):

  # Convolutional_block
  X_copy = X

  f1 , f2, f3 = filter

  # Main Path
  X = Conv2D(f1, (1,1),strides = (1,1), name ='res_'+str(stage)+'_conv_a', kernel_initi
  X = MaxPool2D((2,2))(X)
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_conv_a')(X)
  X = Activation('relu')(X)

  X = Conv2D(f2, kernel_size = (3,3), strides =(1,1), padding = 'same', name ='res_'+st
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_conv_b')(X)
  X = Activation('relu')(X)

  X = Conv2D(f3, kernel_size = (1,1), strides =(1,1),name ='res_'+str(stage)+'_conv_c',
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_conv_c')(X)


  # Short path
  X_copy = Conv2D(f3, kernel_size = (1,1), strides =(1,1),name ='res_'+str(stage)+'_con
  X_copy = MaxPool2D((2,2))(X_copy)
  X_copy = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_conv_copy')(X_copy)

  # ADD
  X = Add()([X,X_copy])
  X = Activation('relu')(X)

  # Identity Block 1
  X_copy = X


  # Main Path
  X = Conv2D(f1, (1,1),strides = (1,1), name ='res_'+str(stage)+'_identity_1_a', kernel
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_1_a')(X)
  X = Activation('relu')(X)

  X = Conv2D(f2, kernel_size = (3,3), strides =(1,1), padding = 'same', name ='res_'+st
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_1_b')(X)
  X = Activation('relu')(X)
```

```
    X = Conv2D(f3, kernel_size = (1,1), strides =(1,1),name ='res_'+str(stage)+'_identity
    X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_1_c')(X)

    # ADD
    X = Add()([X,X_copy])
    X = Activation('relu')(X)

    # Identity Block 2
    X_copy = X


    # Main Path
    X = Conv2D(f1, (1,1),strides = (1,1), name ='res_'+str(stage)+'_identity_2_a', kernel
    X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_2_a')(X)
    X = Activation('relu')(X)

    X = Conv2D(f2, kernel_size = (3,3), strides =(1,1), padding = 'same', name ='res_'+st
    X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_2_b')(X)
    X = Activation('relu')(X)

    X = Conv2D(f3, kernel_size = (1,1), strides =(1,1),name ='res_'+str(stage)+'_identity
    X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_2_c')(X)

    # ADD
    X = Add()([X,X_copy])
    X = Activation('relu')(X)

    return X
```

```
input_shape = (96, 96, 1)

# Input tensor shape
X_input = Input(input_shape)

# Zero-padding
X = ZeroPadding2D((3,3))(X_input)

# 1 - stage
X = Conv2D(64, (7,7), strides= (2,2), name = 'conv1', kernel_initializer= glorot_unifor
X = BatchNormalization(axis =3, name = 'bn_conv1')(X)
X = Activation('relu')(X)
X = MaxPooling2D((3,3), strides= (2,2))(X)

# 2 - stage
X = res_block(X, filter= [64,64,256], stage= 2)

# 3 - stage
X = res_block(X, filter= [128,128,512], stage= 3)


# Average Pooling
X = AveragePooling2D((2,2), name = 'Averagea_Pooling')(X)

# Final layer
```

```
X = Flatten()(X)
X = Dense(4096, activation = 'relu')(X)
X = Dropout(0.2)(X)
X = Dense(2048, activation = 'relu')(X)
X = Dropout(0.1)(X)
X = Dense(30, activation = 'relu')(X)


model_1_facialKeyPoints = Model( inputs= X_input, outputs = X)
model_1_facialKeyPoints.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 96, 96, 1)]       0         
_________________________________________________________________
zero_padding2d (ZeroPadding2D)  (None, 102, 102, 1)    0         
_________________________________________________________________
conv1 (Conv2D)               (None, 48, 48, 64)        3200      
_________________________________________________________________
bn_conv1 (BatchNormalization)   (None, 48, 48, 64)     256       
_________________________________________________________________
activation (Activation)      (None, 48, 48, 64)        0         
_________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 23, 23, 64)     0         
_________________________________________________________________
res_2_conv_a (Conv2D)        (None, 23, 23, 64)        4160      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 11, 11, 64)     0         
_________________________________________________________________
bn_2_conv_a (BatchNormalization (None, 11, 11, 64)     256       
_________________________________________________________________
activation_1 (Activation)    (None, 11, 11, 64)        0         
_________________________________________________________________
res_2_conv_b (Conv2D)        (None, 11, 11, 64)        36928     
_________________________________________________________________
bn_2_conv_b (BatchNormalization (None, 11, 11, 64)     256       
_________________________________________________________________
activation_2 (Activation)    (None, 11, 11, 64)        0         
_________________________________________________________________
res_2_conv_copy (Conv2D)     (None, 23, 23, 256)       16640     
_________________________________________________________________
res_2_conv_c (Conv2D)        (None, 11, 11, 256)       16640     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 11, 11, 256)    0         
_________________________________________________________________
bn_2_conv_c (BatchNormalization (None, 11, 11, 256)    1024      
_________________________________________________________________
bn_2_conv_copy (BatchNormalizat (None, 11, 11, 256)    1024      
_________________________________________________________________
add (Add)                    (None, 11, 11, 256)       0         
                                                                 
_________________________________________________________________
activation_3 (Activation)    (None, 11, 11, 256)       0         
_________________________________________________________________
res_2_identity_1_a (Conv2D)  (None, 11, 11, 64)        16448     
_________________________________________________________________
bn_2_identity_1_a (BatchNormali (None, 11, 11, 64)     256       
_________________________________________________________________
```

```
activation_4 (Activation)       (None, 11, 11, 64)   0           bn_2_identity_1_a
__________________________________________________________________________________
res_2_identity_1_b (Conv2D)     (None, 11, 11, 64)   36928       activation_4[0][0
__________________________________________________________________________________
bn_2_identity_1_b (BatchNormali (None, 11, 11, 64)   256         res_2_identity_1_
__________________________________________________________________________________
activation_5 (Activation)       (None, 11, 11, 64)   0           bn_2_identity_1_b
__________________________________________________________________________________
res_2_identity_1_c (Conv2D)     (None, 11, 11, 256)  16640       activation_5[0][0
```

MINI CHALLENGE 10:

- Experiment with changing the network architecture by removing 2 MaxPooling layers from the Res Block and train the model
- Try to add 'X = res_block(X, filter= [256,256,1024], stage= 4)' Block after stage #3 block.
- What did you observe? Comment on your answer

TASK 10: COMPILE AND TRAIN KEY FACIAL POINTS DETECTION DEEP LEARNING MODEL

```
adam = tf.keras.optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999, a
model_1_facialKeyPoints.compile(loss = "mean_squared_error", optimizer = adam , metrics
# Check this out for more information on Adam optimizer: https://www.tensorflow.org/api
```

```
# save the best model with least validation loss
checkpointer = ModelCheckpoint(filepath = "FacialKeyPoints_weights.hdf5", verbose = 1,
```

```
history = model_1_facialKeyPoints.fit(X_train, y_train, batch_size = 32, epochs = 2, va
```

```
Epoch 1/2
153/153 [==============================] - 41s 59ms/step - loss: 608.3210 - accuracy

Epoch 00001: val_loss improved from inf to 933.82471, saving model to FacialKeyPoints
Epoch 2/2
153/153 [==============================] - 8s 51ms/step - loss: 111.2732 - accuracy:

Epoch 00002: val_loss improved from 933.82471 to 308.11569, saving model to FacialKey
```

```
# save the model architecture to json file for future use

model_json = model_1_facialKeyPoints.to_json()
with open("FacialKeyPoints-model.json","w") as json_file:
  json_file.write(model_json)

```

MINI CHALLENGE #11:

- Experiment with changing the batch size and validation split value and retrain the model (Take Home)

TASK 11: ASSESS TRAINED KEY FACIAL POINTS DETECTION MODEL PERFORMANCE

```
with open('detection.json', 'r') as json_file:
    json_savedModel= json_file.read()

# load the model architecture
model_1_facialKeyPoints = tf.keras.models.model_from_json(json_savedModel)
model_1_facialKeyPoints.load_weights('weights_keypoint.hdf5')
adam = tf.keras.optimizers.Adam(learning_rate=0.0001, beta_1=0.9, beta_2=0.999, amsgrad
model_1_facialKeyPoints.compile(loss="mean_squared_error", optimizer= adam , metrics =
```

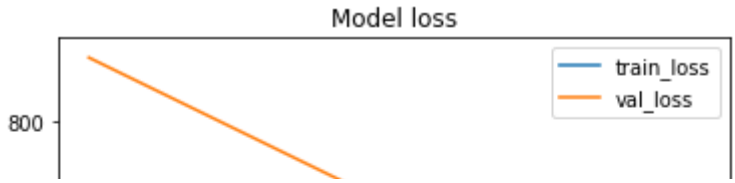
```
# Evaluate the model

result = model_1_facialKeyPoints.evaluate(X_test, y_test)
print("Accuracy : {}".format(result[1]))
```

```
41/41 [==============================] - 2s 17ms/step - loss: 3.5929 - accuracy: 0.85
Accuracy : 0.8442367315292358
```

```
# Get the model keys
history.history.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

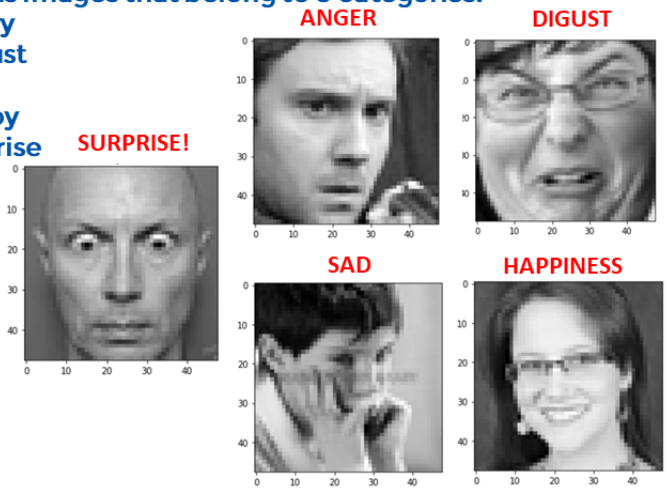
```
# Plot the training artifacts

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train_loss','val_loss'], loc = 'upper right')
plt.show()
```



PART 2. FACIAL EXPRESSION DETECTION

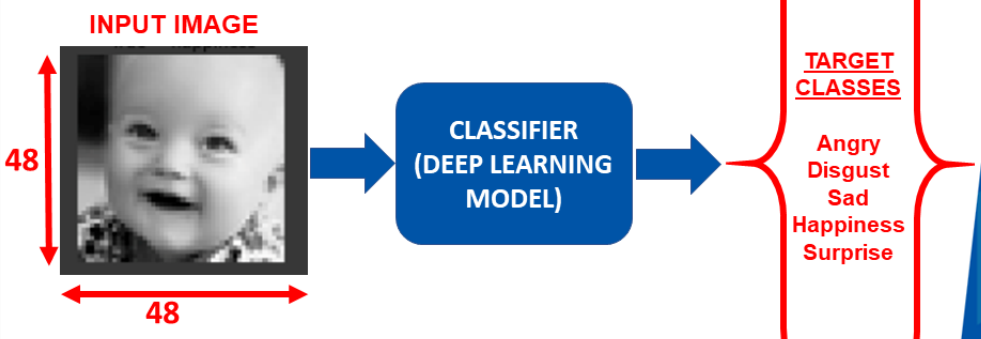
PART 2. FACIAL EXPRESSION (EMOTION) DETECTION

- The second model will classify people's emotion.
- Data contains images that belong to 5 categories:
  - 0 = Angry
  - 1 = Disgust
  - 2 = Sad
  - 3 = Happy
  - 4 = Surprise



Data is source from Kaggle: https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data

PART 2. FACIAL EXPRESSION (EMOTION) DETECTION



TASK 12: IMPORT & EXPLORE DATASET FOR FACIAL EXPRESSION DETECTION

```
1 # read the csv files for the facial expression data
2 facialexpression_df = pd.read_csv('icml_face_data.csv')
```

```
1 facialexpression_df
```

	emotion	pixels
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...
1	0	151 150 147 155 148 133 111 140 170 174 182 15...
2	2	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...
3	2	20 17 19 21 25 38 42 42 46 54 56 62 63 66 82 1...
4	3	77 78 79 79 78 75 60 55 47 48 58 73 77 79 57 5...
...	...	...
24563	3	0 39 81 80 104 97 51 64 68 46 41 67 53 68 70 5...
24564	0	181 177 176 156 178 144 136 132 122 107 131 16...
24565	3	178 174 172 173 181 188 191 194 196 199 200 20...
24566	0	17 17 16 23 28 22 19 17 25 26 20 24 31 19 27 9...
24567	3	30 28 28 29 31 30 42 68 79 81 77 67 67 71 63 6...

```
1 facialexpression_df[' pixels'][0] # String format
```

```
'70 80 82 72 58 58 60 63 54 58 60 48 89 115 121 119 115 110 98 91 84 84 90 99 110 12
6 143 153 158 171 169 172 169 165 129 110 113 107 95 79 66 62 56 57 61 52 43 41 65 6
1 58 57 56 69 75 70 65 56 54 105 146 154 151 151 155 155 150 147 147 148 152 158 164
172 177 182 186 189 188 190 188 180 167 116 95 103 97 77 72 62 55 58 54 56 52 44 50
```

```
1 # function to convert pixel values in string format to array format
2
3 def string2array(x):
4   return np.array(x.split(' ')).reshape(48, 48, 1).astype('float32')
```

```
1 # Resize images from (48, 48) to (96, 96)
2
3 def resize(x):
4
5   img = x.reshape(48, 48)
6   return cv2.resize(img, dsize=(96, 96), interpolation = cv2.INTER_CUBIC)
```

```
1 facialexpression_df[' pixels'] = facialexpression_df[' pixels'].apply(lambda x: string2
```

```
1 facialexpression_df[' pixels'] = facialexpression_df[' pixels'].apply(lambda x: resize(
```

```
1 facialexpression_df.head()
```

	emotion	pixels
0	0	[[69.316925, 73.03865, 79.13719, 84.17186, 85....
1	0	[[151.09435, 150.91393, 150.65791, 148.96367, ...
2	2	[[23.061905, 25.50914, 29.47847, 33.99843, 36....
3	2	[[20.083221, 19.079437, 17.398712, 17.158691, ...
4	3	[[76.26172, 76.54747, 77.001785, 77.7672, 78.4...

```
1 # check the shape of data_frame
2 facialexpression_df.shape
```

```
(24568, 2)
```

```
1 # check for the presence of null values in the data frame
2 facialexpression_df.isnull().sum()
```

```
emotion    0
 pixels    0
dtype: int64
```

```
1 label_to_text = {0:'anger', 1:'disgust', 2:'sad', 3:'happiness', 4: 'surprise'}
```

MINI CHALLENGE 12:

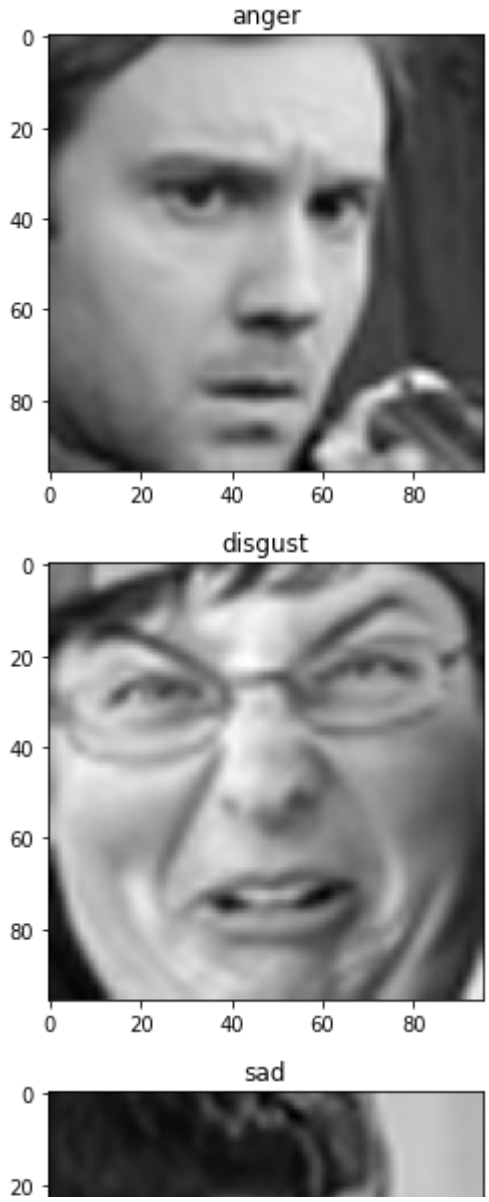
- Visualize the first image in the dataframe and make sure that the image is not distorted by resizing or reshaping operations

```
1 plt.imshow(facialexpression_df[' pixels'][0], cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x7ff020a65dd0>
```

TASK 13: VISUALIZE IMAGES AND PLOT LABELS

```
emotions = [0, 1, 2, 3, 4]

for i in emotions:
  data = facialexpression_df[facialexpression_df['emotion'] == i][:1]
  img = data[' pixels'].item()
  img = img.reshape(96, 96)
  plt.figure()
  plt.title(label_to_text[i])
  plt.imshow(img, cmap = 'gray')
```



MINI CHALLENGE 13:

- Plot bar chart to outline how many samples (images) are present per emotion

```
1 facialexpression_df.emotion.value_counts().index
```

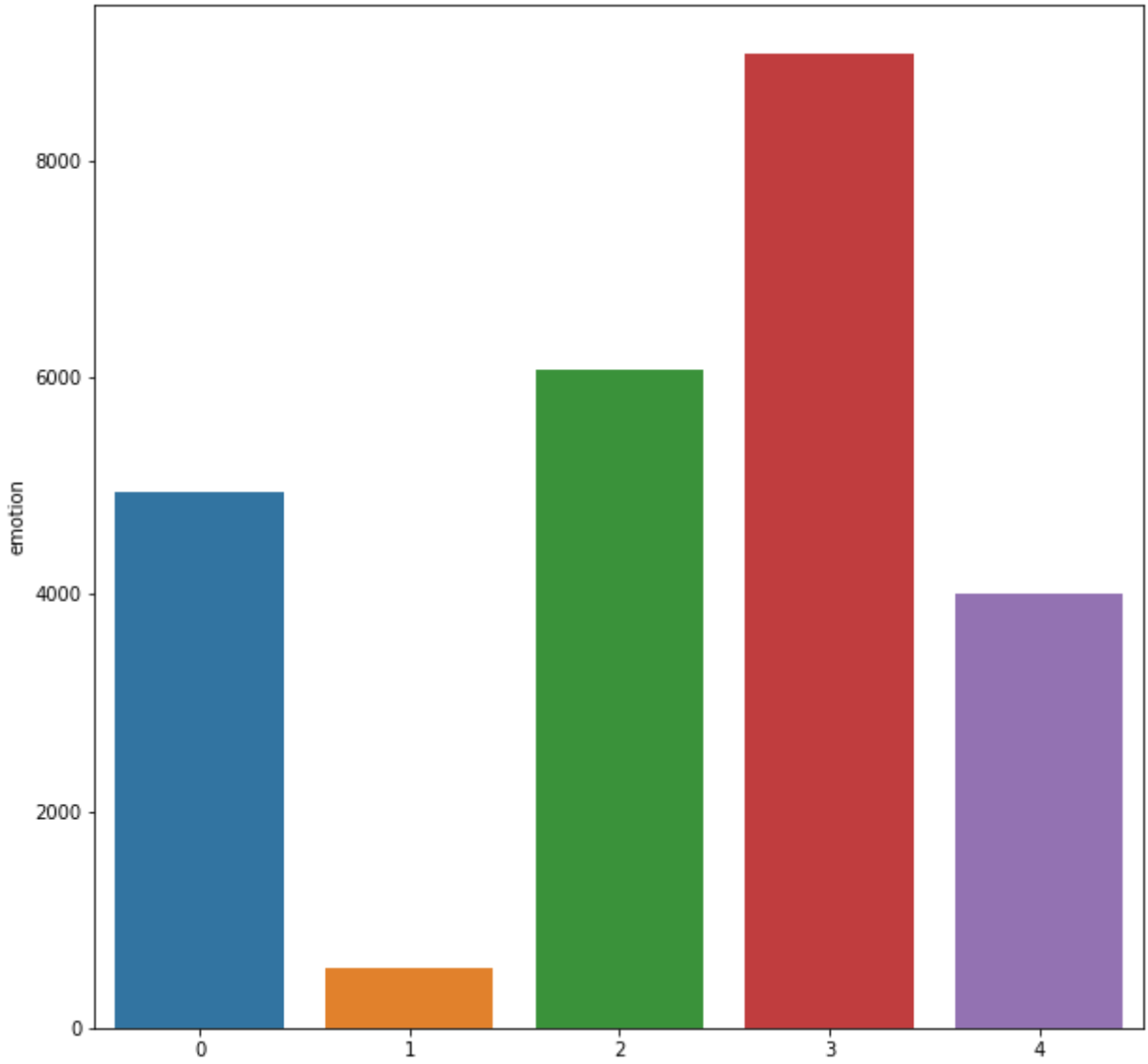
```
Int64Index([3, 2, 0, 4, 1], dtype='int64')
```

```
1 facialexpression_df.emotion.value_counts()
```

```
3    8989
2    6077
0    4953
4    4002
1     547
Name: emotion, dtype: int64
```

```
1 plt.figure(figsize = (10,10))
2 sns.barplot(x = facialexpression_df.emotion.value_counts().index, y = facialexpression_
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff020e1f650>
```



TASK 14: PERFORM DATA PREPARATION AND IMAGE AUGMENTATION

```
# split the dataframe in to features and labels
from keras.utils import to_categorical

X = facialexpression_df[' pixels']
y = to_categorical(facialexpression_df['emotion'])
```

```
X[0]
```

```
array([[ 69.316925,  73.03865 ,  79.13719 , ...,  40.73619 ,  40.6584  ,
         40.5394  ],
       [ 68.66325 ,  70.95224 ,  74.71768 , ...,  43.274765,  42.037186,
         41.223618],
       [ 67.768814,  67.71646 ,  67.67172 , ...,  47.39473 ,  44.26329 ,
         42.31465 ],
       ...,
       [ 74.89021 ,  76.79553 ,  79.90259 , ...,  71.463425,  59.32875 ,
         51.74736 ],
```

```
            [ 76.68126 ,  75.808105,  74.25452 , ...,  93.943375,  79.77934 ,
              71.13246 ],
            [ 77.63858 ,  75.15074 ,  70.8846  , ..., 107.66351 ,  92.23212 ,
              82.91585 ]], dtype=float32)
```

```
1 y
```

```
    array([[1., 0., 0., 0., 0.],
           [1., 0., 0., 0., 0.],
           [0., 0., 1., 0., 0.],
           ...,
           [0., 0., 0., 1., 0.],
           [1., 0., 0., 0., 0.],
           [0., 0., 0., 1., 0.]], dtype=float32)
```

```
1 X = np.stack(X, axis = 0)
2 X = X.reshape(24568, 96, 96, 1)
3
4 print(X.shape, y.shape)
```

```
    (24568, 96, 96, 1) (24568, 5)
```

```
1 # split the dataframe in to train, test and validation data frames
2
3 from sklearn.model_selection import train_test_split
4
5 X_train, X_Test, y_train, y_Test = train_test_split(X, y, test_size = 0.1, shuffle = Tr
6 X_val, X_Test, y_val, y_Test = train_test_split(X_Test, y_Test, test_size = 0.5, shuffl
```

```
1 print(X_val.shape, y_val.shape)
```

```
    (1228, 96, 96, 1) (1228, 5)
```

```
1 print(X_Test.shape, y_Test.shape)
```

```
    (1229, 96, 96, 1) (1229, 5)
```

```
1 print(X_train.shape, y_train.shape)
```

```
    (22111, 96, 96, 1) (22111, 5)
```

```
1 # image pre-processing
2
3 X_train = X_train/255
4 X_val   = X_val /255
5 X_Test  = X_Test/255
```

```
1 X_train
```

```
    array([[[[0.3771706 ],
             [0.3790865 ],
             [0.3820961 ],
```

```
       ...,
       [0.45234087],
       [0.44613636],
       [0.4421392 ]],

      [[0.38343853],
       [0.38631067],
       [0.39089522],
       ...,
       [0.4628421 ],
       [0.45598474],
       [0.4515768 ]],

      [[0.393314  ],
       [0.3977934 ],
       [0.4050252 ],
       ...,
       [0.47990957],
       [0.47193322],
       [0.466824  ]],

      ...,

      [[0.703555  ],
       [0.7079123 ],
       [0.7149431 ],
       ...,
       [0.70672727],
       [0.7042064 ],
       [0.70251155]],

      [[0.69810563],
       [0.703704  ],
       [0.71277887],
       ...,
       [0.7021006 ],
       [0.69625074],
       [0.6925468 ]],

      [[0.69464314],
       [0.70101464],
       [0.71136284],
       ...,
       [0.6991957 ],
       [0.6912883 ],
       [0.6863428 ]]],


     [[[0.33348885],
       [0.34494588],
       [0.36367542],
       ...,
       [0.15499023],
       [0.15623857],
       [0.15686275]],
```

```
1 train_datagen = ImageDataGenerator(
2 rotation_range = 15,
3     width_shift_range = 0.1,
4     height_shift_range = 0.1,
```

```
    height_shift_range = 0.1,
    shear_range = 0.1,
    zoom_range = 0.1,
    horizontal_flip = True,
    fill_mode = "nearest")
```

MINI CHALLENGE 14:

- Try at least 2 additional data augmentation techniques

TASK 15: BUILD AND TRAIN DEEP LEARNING MODEL FOR FACIAL EXPRESSION CLASSIFICATION

```
input_shape = (96, 96, 1)

# Input tensor shape
X_input = Input(input_shape)

# Zero-padding
X = ZeroPadding2D((3, 3))(X_input)

# 1 - stage
X = Conv2D(64, (7, 7), strides= (2, 2), name = 'conv1', kernel_initializer= glorot_unif
X = BatchNormalization(axis =3, name = 'bn_conv1')(X)
X = Activation('relu')(X)
X = MaxPooling2D((3, 3), strides= (2, 2))(X)

# 2 - stage
X = res_block(X, filter= [64, 64, 256], stage= 2)

# 3 - stage
X = res_block(X, filter= [128, 128, 512], stage= 3)

# 4 - stage
# X = res_block(X, filter= [256, 256, 1024], stage= 4)

# Average Pooling
X = AveragePooling2D((4, 4), name = 'Averagea_Pooling')(X)

# Final layer
X = Flatten()(X)
X = Dense(5, activation = 'softmax', name = 'Dense_final', kernel_initializer= glorot_u

model_2_emotion = Model( inputs= X_input, outputs = X, name = 'Resnet18')

model_2_emotion.summary()
```

```
Model: "Resnet18"
__________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
```

input_2 (InputLayer)	[(None, 96, 96, 1)]	0	
zero_padding2d_1 (ZeroPadding2D	(None, 102, 102, 1)	0	input_2[0][0]
conv1 (Conv2D)	(None, 48, 48, 64)	3200	zero_padding2d_1[(
bn_conv1 (BatchNormalization)	(None, 48, 48, 64)	256	conv1[0][0]
activation_19 (Activation)	(None, 48, 48, 64)	0	bn_conv1[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 23, 23, 64)	0	activation_19[0][(
res_2_conv_a (Conv2D)	(None, 23, 23, 64)	4160	max_pooling2d_5[0
max_pooling2d_6 (MaxPooling2D)	(None, 11, 11, 64)	0	res_2_conv_a[0][0
bn_2_conv_a (BatchNormalization	(None, 11, 11, 64)	256	max_pooling2d_6[0
activation_20 (Activation)	(None, 11, 11, 64)	0	bn_2_conv_a[0][0]
res_2_conv_b (Conv2D)	(None, 11, 11, 64)	36928	activation_20[0][(
bn_2_conv_b (BatchNormalization	(None, 11, 11, 64)	256	res_2_conv_b[0][0
activation_21 (Activation)	(None, 11, 11, 64)	0	bn_2_conv_b[0][0]
res_2_conv_copy (Conv2D)	(None, 23, 23, 256)	16640	max_pooling2d_5[0
res_2_conv_c (Conv2D)	(None, 11, 11, 256)	16640	activation_21[0][(
max_pooling2d_7 (MaxPooling2D)	(None, 11, 11, 256)	0	res_2_conv_copy[0
bn_2_conv_c (BatchNormalization	(None, 11, 11, 256)	1024	res_2_conv_c[0][0
bn_2_conv_copy (BatchNormalizat	(None, 11, 11, 256)	1024	max_pooling2d_7[0
add_6 (Add)	(None, 11, 11, 256)	0	bn_2_conv_c[0][0] bn_2_conv_copy[0]
activation_22 (Activation)	(None, 11, 11, 256)	0	add_6[0][0]
res_2_identity_1_a (Conv2D)	(None, 11, 11, 64)	16448	activation_22[0][(
bn_2_identity_1_a (BatchNormali	(None, 11, 11, 64)	256	res_2_identity_1_
activation_23 (Activation)	(None, 11, 11, 64)	0	bn_2_identity_1_a
res_2_identity_1_b (Conv2D)	(None, 11, 11, 64)	36928	activation_23[0][(
bn_2_identity_1_b (BatchNormali	(None, 11, 11, 64)	256	res_2_identity_1_
activation_24 (Activation)	(None, 11, 11, 64)	0	bn_2_identity_1_b
res_2_identity_1_c (Conv2D)	(None, 11, 11, 256)	16640	activation_24[0][(

```
1 # train the network
2 model_2_emotion.compile(optimizer = "Adam", loss = "categorical_crossentropy", metrics
```

```
1 # Recall that the first facial key points model was saved as follows: FacialKeyPoints_w
```

```
2
3 # using early stopping to exit training if validation loss is not decreasing even after
4 earlystopping = EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 1, patience
5
6 # save the best model with lower validation loss
7 checkpointer = ModelCheckpoint(filepath = "FacialExpression_weights.hdf5", verbose = 1,
```

```
1 history = model_2_emotion.fit(train_datagen.flow(X_train, y_train, batch_size=64),
2   validation_data=(X_val, y_val), steps_per_epoch=len(X_train) // 64,
3   epochs= 2, callbacks=[checkpointer, earlystopping])
```

```
Epoch 1/2
345/345 [==============================] - 38s 100ms/step - loss: 1.5466 - accuracy:

Epoch 00001: val_loss improved from inf to 1.53256, saving model to FacialExpression_
Epoch 2/2
345/345 [==============================] - 34s 97ms/step - loss: 0.9355 - accuracy: 0

Epoch 00002: val_loss improved from 1.53256 to 1.11236, saving model to FacialExpress
```

```
1 # saving the model architecture to json file for future use
2
3 model_json = model_2_emotion.to_json()
4 with open("FacialExpression-model.json","w") as json_file:
5   json_file.write(model_json)
```

MINI CHALLENGE #15:

- Experiment with various batch size, patience, optimizers, and network architecture to improve network performance (Take home)

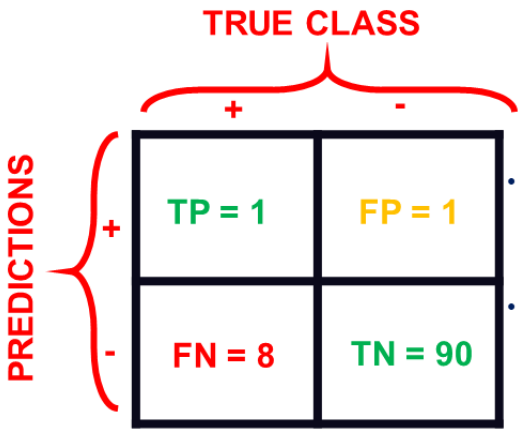
TASK 16: UNDERSTAND HOW TO ASSESS CLASSIFIER MODELS (CONFUSION MATRIX, ACCURACY, PRECISION, AND RECALL)

CONFUSION MATRIX

DEFINITIONS AND KPIS

- A confusion matrix is used to describe the performance of a classification model:
  - True positives (TP): cases when classifier predicted TRUE (they have the disease), and correct class was TRUE (patient has disease).
  - True negatives (TN): cases when model predicted FALSE (no disease), and correct class was FALSE (patient do not have disease).
  - False positives (FP) (Type I error): classifier predicted TRUE, but correct class was FALSE (patient did not have disease).
  - False negatives (FN) (Type II error): classifier predicted FALSE (patient do not have disease), but they actually do have the disease
  - Classification Accuracy = (TP+TN) / (TP + TN + FP + FN)
  - Misclassification rate (Error Rate) = (FP + FN) / (TP + TN + FP + FN)
  - Precision = TP/Total TRUE Predictions = TP/ (TP+FP) (When model predicted TRUE class, how often was it right?)
  - Recall = TP/ Actual TRUE = TP/ (TP+FN) (when the class was actually TRUE, how often did the classifier get it right?)

PRECISION Vs. RECALL EXAMPLE



FACTS:
100 PATIENTS TOTAL
91 PATIENTS ARE HEALTHY
9 PATIENTS HAVE CANCER

- Accuracy is generally misleading and is not enough to assess the performance of a classifier.
- Recall is an important KPI in situations where:
  - Dataset is highly imbalanced; cases when you have small cancer patients compared to healthy ones.

- Classification Accuracy = (TP+TN) / (TP + TN + FP + FN) = 91%
- Precision = TP/Total TRUE Predictions = TP/ (TP+FP) = ½=50%
- Recall = TP/ Actual TRUE = TP/ (TP+FN) = 1/9 = 11%

TASK 17: ASSESS THE PERFORMANCE OF TRAINED FACIAL EXPRESSION CLASSIFIER MODEL

```
with open('emotion.json', 'r') as json_file:
    json_savedModel= json_file.read()

# load the model architecture
model_2_emotion = tf.keras.models.model_from_json(json_savedModel)
model_2_emotion.load_weights('weights_emotions.hdf5')
model_2_emotion.compile(optimizer = "Adam", loss = "categorical_crossentropy", metrics
```

```
score = model_2_emotion.evaluate(X_Test, y_Test)
print('Test Accuracy: {}'.format(score[1]))
```

```
39/39 [==============================] - 2s 19ms/step - loss: 0.3141 - accuracy: 0.88
Test Accuracy: 0.8803905844688416
```

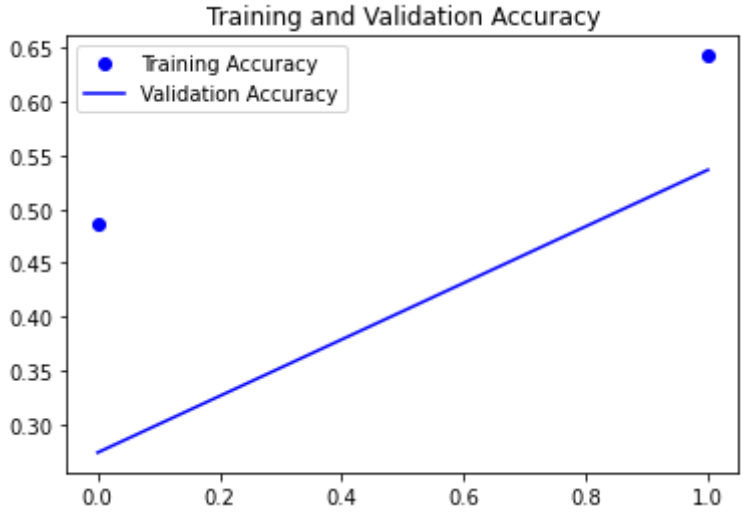
```
history.history.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
epochs = range(len(accuracy))

plt.plot(epochs, accuracy, 'bo', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
```

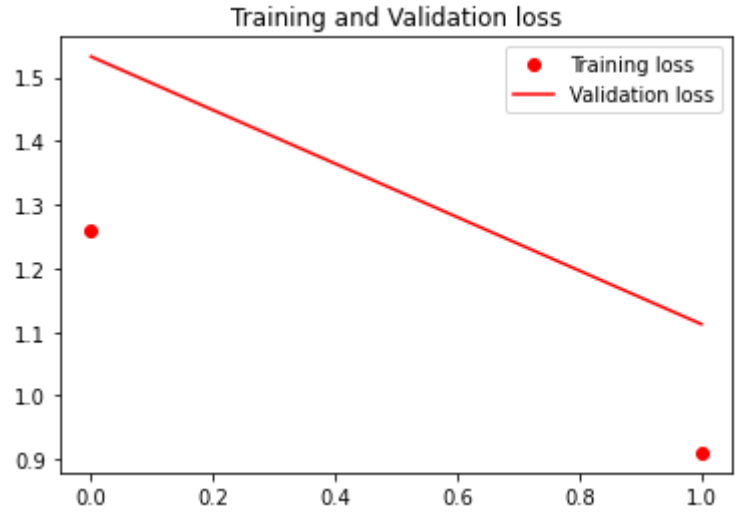
```
<matplotlib.legend.Legend at 0x7ff01905ecd0>
```




```
plt.plot(epochs, loss, 'ro', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
```

```
3 plt.title('Training and Validation loss')
4 plt.legend()
```

```
<matplotlib.legend.Legend at 0x7ff018fdded0>
```



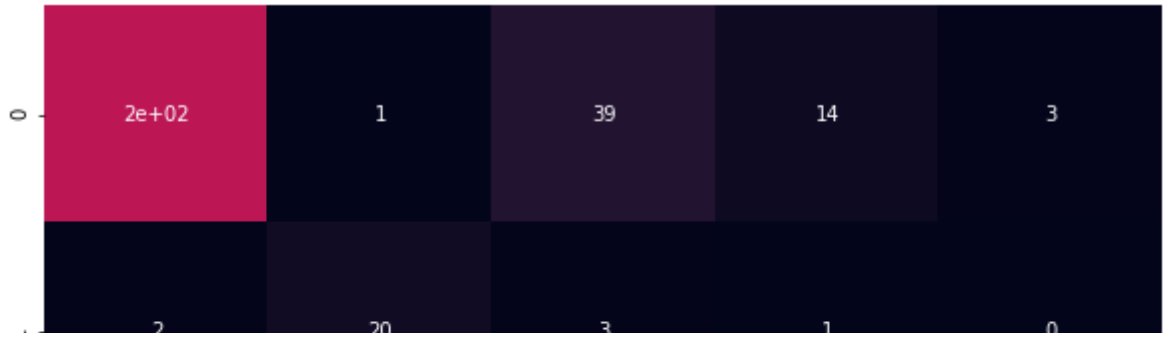

```
1 # predicted_classes = model.predict_classes(X_test)
2 predicted_classes = np.argmax(model_2_emotion.predict(X_Test), axis=-1)
3 y_true = np.argmax(y_Test, axis=-1)
```

```
1 y_true.shape
```

```
(1229,)
```

```
1 from sklearn.metrics import confusion_matrix
2 cm = confusion_matrix(y_true, predicted_classes)
3 plt.figure(figsize = (10, 10))
4 sns.heatmap(cm, annot = True, cbar = False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff018f12410>
```



MINI CHALLENGE 16:

- Print out a grid of 100 images along with their predicted/true label
- Print out the classification report and analyze precision and recall

```
L = 10
W = 10

fig, axes = plt.subplots(L, W, figsize = (24, 24))
axes = axes.ravel()

for i in np.arange(0, L*W):
    axes[i].imshow(X_test[i].reshape(96,96), cmap = 'gray')
    axes[i].set_title('Prediction = {}\n True = {}'.format(label_to_text[predicted_clas
    axes[i].axis('off')

plt.subplots_adjust(wspace = 1)
```


```
1 from sklearn.metrics import classification_report
2 print(classification_report(y_true, predicted_classes))
```

```
              precision    recall  f1-score   support

           0       0.83      0.78      0.81       259
           1       0.95      0.77      0.85        26
           2       0.82      0.87      0.84       300
           3       0.92      0.95      0.93       455
           4       0.96      0.88      0.92       189
```

```
              precision    recall  f1-score   support

    accuracy                         0.88      1229
   macro avg       0.90      0.85      0.87      1229
weighted avg       0.88      0.88      0.88      1229
```

PART 3. COMBINE BOTH FACIAL EXPRESSION AND KEY POINTS DETECTION MODELS

TASK 18: COMBINE BOTH MODELS (1) FACIAL KEY POINTS DETECTION AND (2) FACIAL EXPRESSION MODELS

```
def predict(X_test):

  # Making prediction from the keypoint model
  df_predict = model_1_facialKeyPoints.predict(X_test)

  # Making prediction from the emotion model
  df_emotion = np.argmax(model_2_emotion.predict(X_test), axis=-1)

  # Reshaping array from (856,) to (856,1)
  df_emotion = np.expand_dims(df_emotion, axis = 1)

  # Converting the predictions into a dataframe
  df_predict = pd.DataFrame(df_predict, columns= columns)

  # Adding emotion into the predicted dataframe
  df_predict['emotion'] = df_emotion

  return df_predict
```

```
df_predict = predict(X_test)
```

```
df_predict.head()
```

	left_eye_center_x	left_eye_center_y	right_eye_center_x	right_eye_center_y	lef
0	64.338631	34.892876	29.859175	35.263588	
1	34.330132	34.536922	68.024673	41.444901	
2	39.074493	38.337498	66.981659	36.173569	
3	67.240051	37.149734	30.336187	34.132309	
4	66.721527	38.846989	30.033785	36.100227	

MINI CHALLENGE 17:

- Plot a grid of 16 images along with their predicted emotion and facial key points

```
# Plotting the test images and their predicted keypoints and emotions

fig, axes = plt.subplots(4, 4, figsize = (24, 24))
axes = axes.ravel()

for i in range(16):

    axes[i].imshow(X_test[i].squeeze(),cmap='gray')
    axes[i].set_title('Prediction = {}'.format(label_to_text[df_predict['emotion'][i]])
    axes[i].axis('off')
    for j in range(1,31,2):
            axes[i].plot(df_predict.loc[i][j-1], df_predict.loc[i][j], 'rx')
```

PART 4. DEPLOY BOTH TRAINED MODELS

TASK 19. SAVE THE TRAINED MODEL FOR DEPLOYMENT

MODEL DEPLOYMENT USING TENSORFLOW SERVING:

- Let's assume that we already trained our model and it is generating good results on the testing data.
- Now, we want to integrate our trained Tensorflow model into a web app and deploy the model in production level environment.
- The following objective can be obtained using TensorFlow Serving. TensorFlow Serving is a high-performance serving system for machine learning models, designed for production environments.
- With the help of TensorFlow Serving, we can easily deploy new

RUNNING TENSORFLOW SERVING:

- There are some important parameters:
  - rest_api_port: The port that you'll use for REST requests.
  - model_name: You'll use this in the URL of REST requests. You can choose any name
  - model_base_path: This is the path to the directory where you've saved your model.
- For more information regarding REST, check this out: https://www.codecademy.com/articles/what-is-rest
- REST is a revival of HTTP in which http commands have semantic meaning.

RUNNING TENSORFLOW SERVING:

- There are some important parameters:
  - rest_api_port: The port that you'll use for REST requests.
  - model_name: You'll use this in the URL of REST requests. You can choose any name
  - model_base_path: This is the path to the directory where you've saved your model.
- For more information regarding REST, check this out: https://www.codecademy.com/articles/what-is-rest
- REST is a revival of HTTP in which http commands have semantic meaning.

MAKING REQUEST IN TENSORFLOW SERVING:

- In-order to make prediction using TensorFlow Serving, we need to pass the inference requests (image data) as a JSON object.
- Then, we use python requests library to make a post request to the deployed model, by passing in the JSON object containing inference requests (image data).
- Finally, we get the prediction from the post request made to the deployed model and then use argmax function to find the predicted class.

- We now need to save our trained model and it has to be saved in a SavedModel format.
- The model will have a version number and will be saved in a structured directory
- tf.saved_model.save is a function used to build a saved model that is suitable for serving using Tensorflow Serving.
- After the model is saved, we can now use TensorFlow Serving to start making inference requests using a specific version of our trained model "servable".
- Use SavedModel to save and load your model—variables, the graph, and the graph's metadata.
- Check this out for more information: https://www.tensorflow.org/guide/saved_model

```
import json
import tensorflow.keras.backend as K

def deploy(directory, model):
  MODEL_DIR = directory
  version = 1

```

```
# Let's join the temp model directory with our chosen version number
# The expected result will be = '\tmp\version number'
export_path = os.path.join(MODEL_DIR, str(version))
print('export_path = {}\n'.format(export_path))

# Let's save the model using saved_model.save
# If the directory already exists, we will remove it using '!rm'
# rm removes each file specified on the command line.

if os.path.isdir(export_path):
  print('\nAlready saved a model, cleaning up\n')
  !rm -r {export_path}

tf.saved_model.save(model, export_path)

os.environ["MODEL_DIR"] = MODEL_DIR
```

TASK 20. SERVE THE MODEL USING TENSORFLOW SERVING

```
# Let's add tensorflow-model-server package to our list of packages
!echo "deb http://storage.googleapis.com/tensorflow-serving-apt stable tensorflow-model
curl https://storage.googleapis.com/tensorflow-serving-apt/tensorflow-serving.release.p
!apt update
```

```
deb http://storage.googleapis.com/tensorflow-serving-apt stable tensorflow-model-serv
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2943  100  2943    0     0  14081      0 --:--:-- --:--:-- --:--:-- 14014
OK
Get:1 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ InRelease [3,626 B]
Get:2 http://storage.googleapis.com/tensorflow-serving-apt stable InRelease [3,012 B]
Ign:3 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  InR
Get:4 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ Packages [52.7 kB]
Get:5 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic InRelease [15.9 kB]
Get:6 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:7 http://archive.ubuntu.com/ubuntu bionic InRelease
Ign:8 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804
Hit:9 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Rel
Hit:10 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu18
Get:11 http://storage.googleapis.com/tensorflow-serving-apt stable/tensorflow-model-s
Get:12 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:13 http://storage.googleapis.com/tensorflow-serving-apt stable/tensorflow-model-s
Hit:16 http://ppa.launchpad.net/cran/libgit2/ubuntu bionic InRelease
Get:17 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:18 http://ppa.launchpad.net/deadsnakes/ppa/ubuntu bionic InRelease [15.9 kB]
Get:19 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [2,045 k
Get:20 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [2,172
Hit:21 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic InRelease
Get:22 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main Sources [1,74
Get:23 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [1,4
Get:24 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [2,476 kB]
Get:25 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main amd64 Package
Get:26 http://ppa.launchpad.net/deadsnakes/ppa/ubuntu bionic/main amd64 Packages [39
Fetched 11.1 MB in 7s (1,580 kB/s)
Reading package lists... Done
```

```
Building dependency tree
Reading state information... Done
51 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

```
# Let's install tensorflow model server
!apt-get install tensorflow-model-server
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  tensorflow-model-server
0 upgraded, 1 newly installed, 0 to remove and 51 not upgraded.
Need to get 223 MB of archives.
After this operation, 0 B of additional disk space will be used.
Get:1 http://storage.googleapis.com/tensorflow-serving-apt stable/tensorflow-model-se
Fetched 223 MB in 6s (39.6 MB/s)
Selecting previously unselected package tensorflow-model-server.
(Reading database ... 160980 files and directories currently installed.)
Preparing to unpack .../tensorflow-model-server_2.4.1_all.deb ...
Unpacking tensorflow-model-server (2.4.1) ...
Setting up tensorflow-model-server (2.4.1) ...
```

```
# Let's run TensorFlow serving
```

- We will load our model and start making inferences (predictions) based on it.
- There are some important parameters:
  - rest_api_port: The port that you'll use for REST requests.
  - model_name: You'll use this in the URL of REST requests. You can choose any name
  - model_base_path: This is the path to the directory where you've saved your model.
- For more information regarding REST, check this out: https://www.codecademy.com/articles/what-is-rest
- REST is a revival of HTTP in which http commands have semantic meaning.

```
deploy('/model', model_1_facialKeyPoints)
```

```
export_path = /model/1

INFO:tensorflow:Assets written to: /model/1/assets
```

```
%%bash --bg
nohup tensorflow_model_server \
  --rest_api_port=4500 \
  --model_name=keypoint_model \
  --model_base_path="${MODEL_DIR}" >server.log 2>&1
```

```
Starting job # 0 in a separate thread.
```

```
1 !tail server.log
```

```
2021-04-06 06:19:05.543821: W external/org_tensorflow/tensorflow/core/framework/cpu_a
2021-04-06 06:19:05.570102: W external/org_tensorflow/tensorflow/core/framework/cpu_a
2021-04-06 06:19:05.600111: I external/org_tensorflow/tensorflow/cc/saved_model/loade
2021-04-06 06:19:05.643979: I external/org_tensorflow/tensorflow/cc/saved_model/loade
2021-04-06 06:19:05.654633: I tensorflow_serving/servables/tensorflow/saved_model_war
2021-04-06 06:19:05.654895: I tensorflow_serving/core/loader_harness.cc:87] Successfu
2021-04-06 06:19:05.656928: I tensorflow_serving/model_servers/server.cc:371] Running
[warn] getaddrinfo: address family for nodename not supported
2021-04-06 06:19:05.657973: I tensorflow_serving/model_servers/server.cc:391] Exporti
[evhttp_server.cc : 238] NET_LOG: Entering the event loop ...
```

```
1 deploy('/model1', model_2_emotion)
```

```
export_path = /model1/1

INFO:tensorflow:Assets written to: /model1/1/assets
```

```
1 %%bash --bg
2 nohup tensorflow_model_server \
3   --rest_api_port=4000 \
4   --model_name=emotion_model \
5   --model_base_path="${MODEL_DIR}" >server.log 2>&1
```

```
Starting job # 2 in a separate thread.
```

```
1 !tail server.log
```

```
2021-04-06 06:19:46.928553: I external/org_tensorflow/tensorflow/cc/saved_model/loade
2021-04-06 06:19:46.935572: I external/org_tensorflow/tensorflow/core/platform/profil
2021-04-06 06:19:47.070882: I external/org_tensorflow/tensorflow/cc/saved_model/loade
2021-04-06 06:19:47.112911: I external/org_tensorflow/tensorflow/cc/saved_model/loade
2021-04-06 06:19:47.123440: I tensorflow_serving/servables/tensorflow/saved_model_war
2021-04-06 06:19:47.123777: I tensorflow_serving/core/loader_harness.cc:87] Successfu
2021-04-06 06:19:47.125903: I tensorflow_serving/model_servers/server.cc:371] Running
[warn] getaddrinfo: address family for nodename not supported
2021-04-06 06:19:47.126903: I tensorflow_serving/model_servers/server.cc:391] Exporti
[evhttp_server.cc : 238] NET_LOG: Entering the event loop ...
```

- **Congratulations! now we have successfully loaded a servable version of our model {name: keypoint_model version: 1}**
- **Congratulations! now we have successfully loaded a servable version of our model {name: emotion_model version: 1}**

TASK 21: MAKE REQUESTS TO MODEL IN TENSORFLOW SERVING

```
1 import json
2
```

```
# Let's create a JSON object and make 3 inference requests
data = json.dumps({"signature_name": "serving_default", "instances": X_test[0:3].tolist
print('Data: {} ... {}'.format(data[:50], data[len(data)-52:]))
```

```
Data: {"signature_name": "serving_default", "instances": ... 7], [0.2901960909366607
```

```
!pip install -q requests
```

```
import requests

# Function to make predictions from deployed models
def response(data):
  headers = {"content-type": "application/json"}
  json_response = requests.post('http://localhost:4500/v1/models/keypoint_model/version
  df_predict = json.loads(json_response.text)['predictions']
  json_response = requests.post('http://localhost:4000/v1/models/emotion_model/versions
  df_emotion = np.argmax(json.loads(json_response.text)['predictions'], axis = 1)

  # Reshaping array from (856,) to (856,1)
  df_emotion = np.expand_dims(df_emotion, axis = 1)

  # Converting the predictions into a dataframe
  df_predict= pd.DataFrame(df_predict, columns = columns)

  # Adding emotion into the predicted dataframe
  df_predict['emotion'] = df_emotion

  return df_predict
```

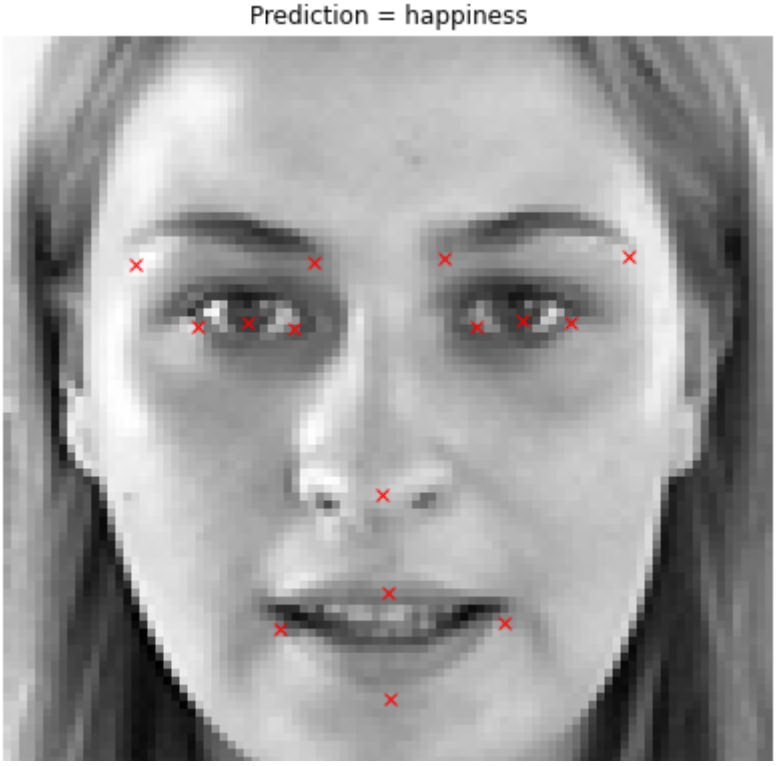
```
# making prediction
df_predict = response(data)
```

```
df_predict
```

	left_eye_center_x	left_eye_center_y	right_eye_center_x	right_eye_center_y	lef
0	64.338638	34.892876	29.859175	35.263588	
1	34.330139	34.536919	68.024681	41.444900	
2	39.074482	38.337521	66.981651	36.173572	

```
# Plotting the test images and their predicted keypoints and emotions

fig, axes = plt.subplots(3, 1, figsize = (24, 24))
axes = axes.ravel()

for i in range(3):

    axes[i].imshow(X_test[i].squeeze(),cmap='gray')
    axes[i].set_title('Prediction = {}'.format(label_to_text[df_predict['emotion'][i]])
```

```
    axes[i].axis('off')
    for j in range(1,31,2):
            axes[i].plot(df_predict.loc[i][j-1], df_predict.loc[i][j], 'rx')
```



MINI CHALLENGE AND IT'S SOLUTIONS

MINI CHALLENGE #1:

- Obtain the average, minimum and maximum values for 'right_eye_center_x'

```
1 keyfacial_df.describe()
```

	left_eye_center_x	left_eye_center_y	right_eye_center_x	right_eye_center_y
count	2140.000000	2140.000000	2140.000000	2140.000000
mean	66.221549	36.842274	29.640269	37.063815
std	2.087683	2.294027	2.051575	2.234334
min	47.835757	23.832996	18.922611	24.773072
25%	65.046300	35.468842	28.472224	35.818377
50%	66.129065	36.913319	29.655440	37.048085
75%	67.332093	38.286438	30.858673	38.333884
max	78.013082	46.132421	42.495172	45.980981

MINI CHALLENGE #2:

- Perform a sanity check on on the data by randomly visualizing 64 new images along with their cooresponding key points

```
import random
# Let's view more images in a grid format
fig = plt.figure(figsize=(20, 20))

for i in range(64):
    k = random.randint(1, len(keyfacial_df))
    ax = fig.add_subplot(8, 8, i + 1)
    image = plt.imshow(keyfacial_df['Image'][k],cmap = 'gray')
    for j in range(1,31,2):
        plt.plot(keyfacial_df.loc[k][j-1], keyfacial_df.loc[k][j], 'rx')

```

MINI CHALLENGE #3:

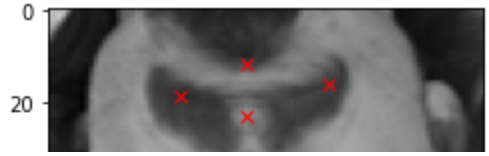
- Augment images by flipping them vertically (Hint: Flip along x-axis and note that if we are flipping along x-axis, x co-ordinates won't change)

```
keyfacial_df_copy = copy.copy(keyfacial_df)

# Flip the images along x axis
keyfacial_df_copy['Image'] = keyfacial_df_copy['Image'].apply(lambda x: np.flip(x, axis

# since we are flipping vertically, x coordinate values would be the same
for i in range(len(columns)):
  if i%2 == 1:
    keyfacial_df_copy[columns[i]] = keyfacial_df_copy[columns[i]].apply(lambda x: 96. -
```

MINI CHALLENGE #4:

- Perform a sanity check and visualize sample images

```
# Show the vertically flipped image
plt.imshow(keyfacial_df_copy['Image'][0],cmap='gray')
for j in range(1, 31, 2):
        plt.plot(keyfacial_df_copy.loc[0][j-1], keyfacial_df_copy.loc[0][j], 'rx')
```



MINI CHALLENGE #5:

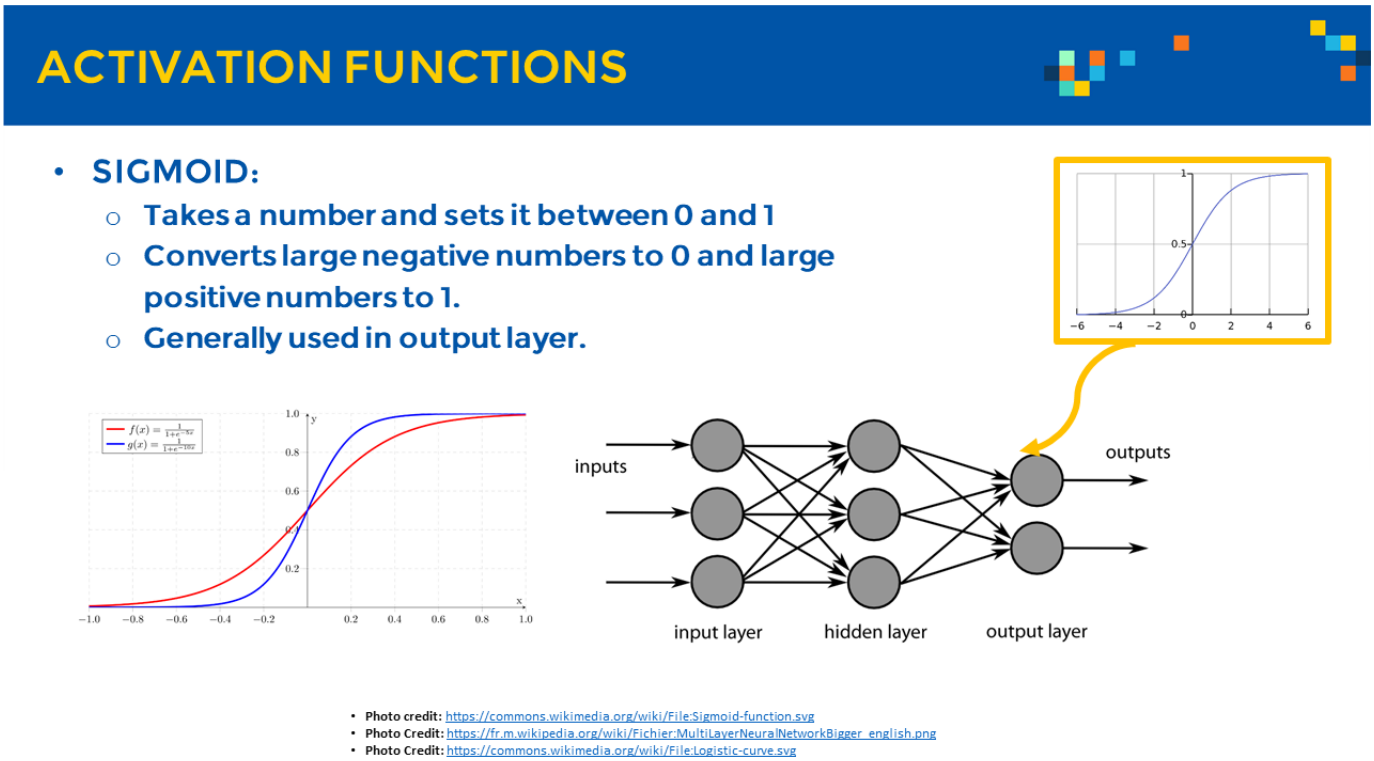
- Try a different value for 'test_size' and verify that the split was successful

```
1 print('Train size =', X_train.shape)
2 print('Test size =', X_test.shape)
```

```
Train size = (22111, 96, 96, 1)
Test size = (1284, 96, 96, 1)
```

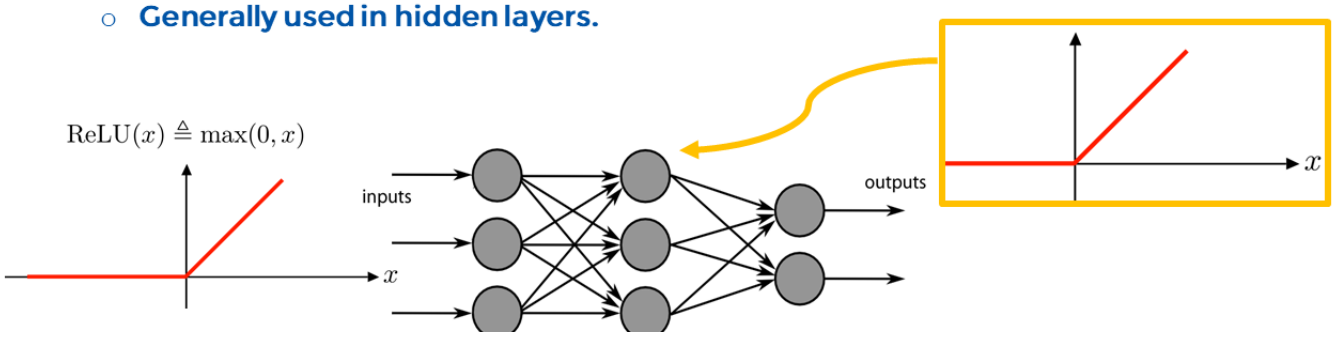
MINI CHALLENGE #6:

- List at least 3 types of activation functions, which type is preferred in the hidden layers?
- List at least 3 different neural networks architecutres and their applications.



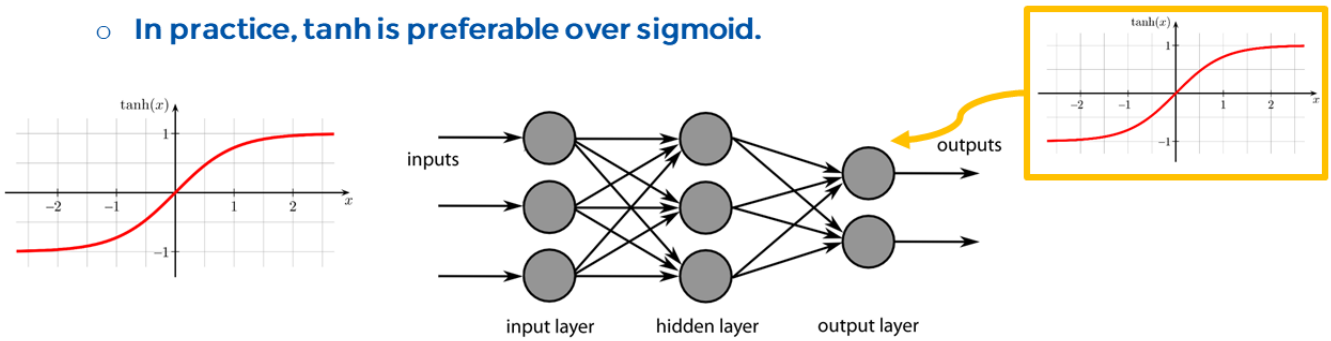
ACTIVATION FUNCTIONS

- **RELU (RECTIFIED LINEAR UNITS):**
  - if input x < 0, output is 0 and if x > 0 the output is x.
  - RELU does not saturate so it avoids vanishing gradient problem.
  - It uses simple thresholding so it is computationally efficient.
  - Generally used in hidden layers.



ACTIVATION FUNCTIONS

- **HYPERBOLIC TANGENT ACTIVATION FUNCTION:**
  - "Tanh" is similar to sigmoid, converts number between -1 and 1.
  - Unlike sigmoid, tanh outputs are zero-centered (range: -1 and 1).
  - Tanh suffers from vanishing gradient problem so it kills gradients when saturated.
  - In practice, tanh is preferable over sigmoid.



- **Photo credit:** https://commons.wikimedia.org/wiki/File:Hyperbolic_Tangent.svg
- **Photo Credit:** https://fr.m.wikipedia.org/wiki/Fichier:MultiLayerNeuralNetworkBigger_english.png

MINI CHALLENGE #7:

- List 3 different neural networks architectures and their applications.

Great Resource by Aravind Pai: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/

MINI CHALLENGE #8:

- What happen when you set the learning rate to the extremes (i.e.: very small value and very large value)? How can you achieve the best of both worlds?

Great Resource by Jason Brownlee: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

MINI CHALLENGE #9:

- What's the performance of ResNets compared to AlexNet in ImageNet dataset?

https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

MINI CHALLENGE #10:

- Experiment with changing the network architecture by removing 2 MaxPooling layers from the Res Block and train the model
- Try to add 'X = res_block(X, filter= [256,256,1024], stage= 4)' Block after stage #3 block.
- What did you observe? Comment on your answer

Number of trainable parameters increase significantly

MINI CHALLENGE #11:

- Experiment with changing the batch size and validation split value and retrain the model (Take Home)

Take home! experiment with them and see if they improve the performance.

MINI CHALLENGE #12:

- Visualize the first image in the dataframe and make sure that the image is not distorted

```
1 plt.imshow(facialexpression_df[' pixels'][0], cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x7ff0d48e2390>
```

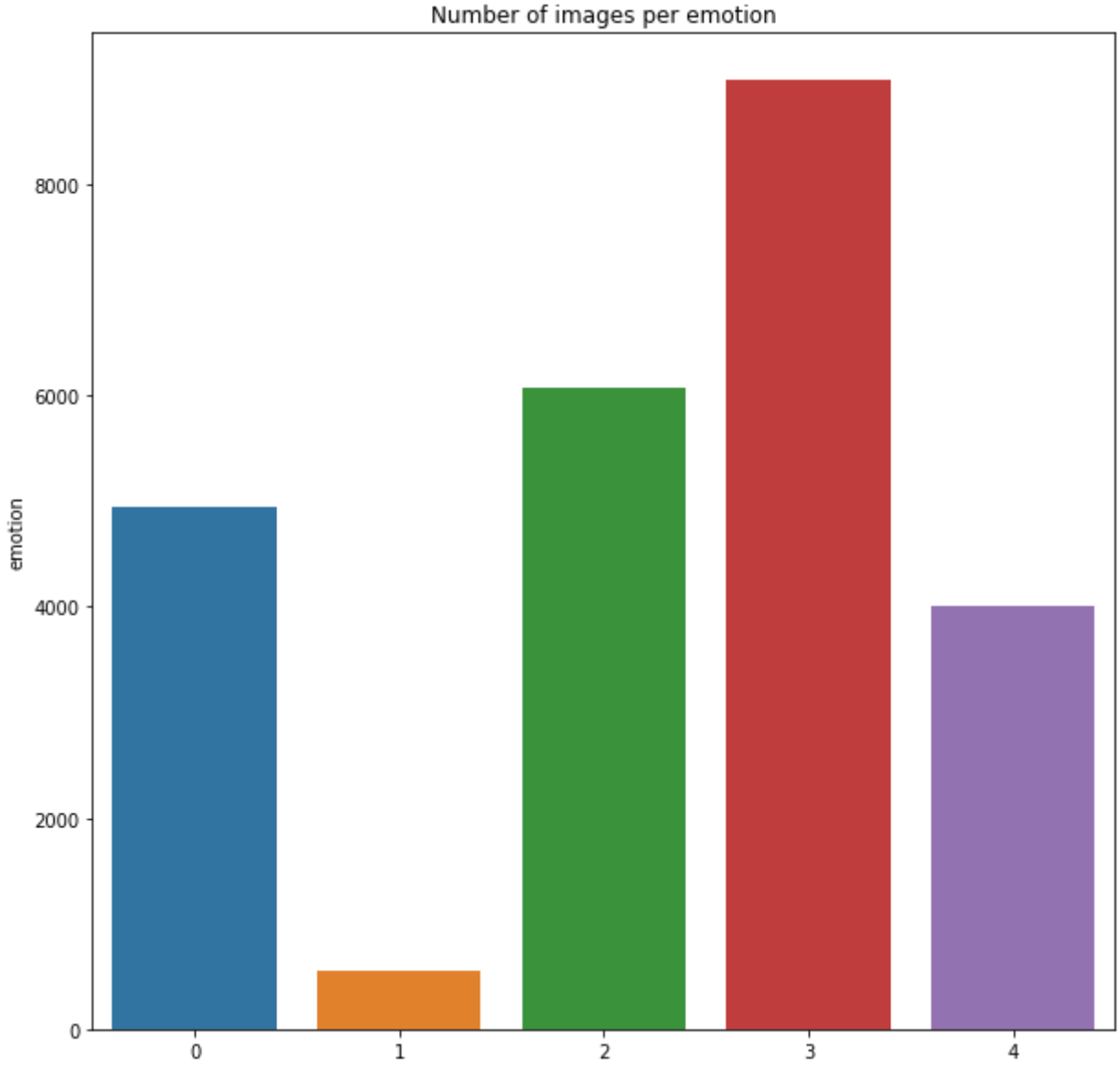
MINI CHALLENGE #13:

- Plot bar chart to outline how many samples (images) are present per emotion

0 20 40 60 80

```
1 plt.figure(figsize=(10,10))
2 sns.barplot(x = facialexpression_df.emotion.value_counts().index, y = facialexpression_
3 plt.title('Number of images per emotion')
```

```
Text(0.5, 1.0, 'Number of images per emotion')
```



MINI CHALLENGE #14:

- Try at least 2 additional data augmentation techniques

Check Keras documentation: https://keras.io/api/preprocessing/image/

```
train_datagen = ImageDataGenerator(
rotation_range = 15,
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    shear_range = 0.1,
    zoom_range = 0.1,
    horizontal_flip = True,
    vertical_flip = True,
    brightness_range = [1.1, 1.5],
    fill_mode = "nearest")

# Note on "Brightness_range"
# 1.0 does not affect image brightness
# numbers less than 1.0 darken the image [0.5, 1.0]
# numbers larger than 1.0 brighten the image [1.0, 1.5]

```

MINI CHALLENGE #15:

- Experiment with various batch size, patience, optimizers, and network architecture to improve network performance (Take home)

Take home! experiment and see if they improve the performance.

MINI CHALLENGE #16:

- Print out a grid of 25 images along with their predicted/true label
- Print out the classification report and analyze precision and recall

```
L = 5
W = 5

fig, axes = plt.subplots(L, W, figsize = (24, 24))
axes = axes.ravel()

for i in np.arange(0, L*W):
    axes[i].imshow(X_test[i].reshape(96,96), cmap = 'gray')
    axes[i].set_title('Prediction = {}\n True = {}'.format(label_to_text[predicted_clas
    axes[i].axis('off')

plt.subplots_adjust(wspace = 1)
```