TASK #1: PROJECT OVERVIEW

KEY FACIAL POINTS DETECTION

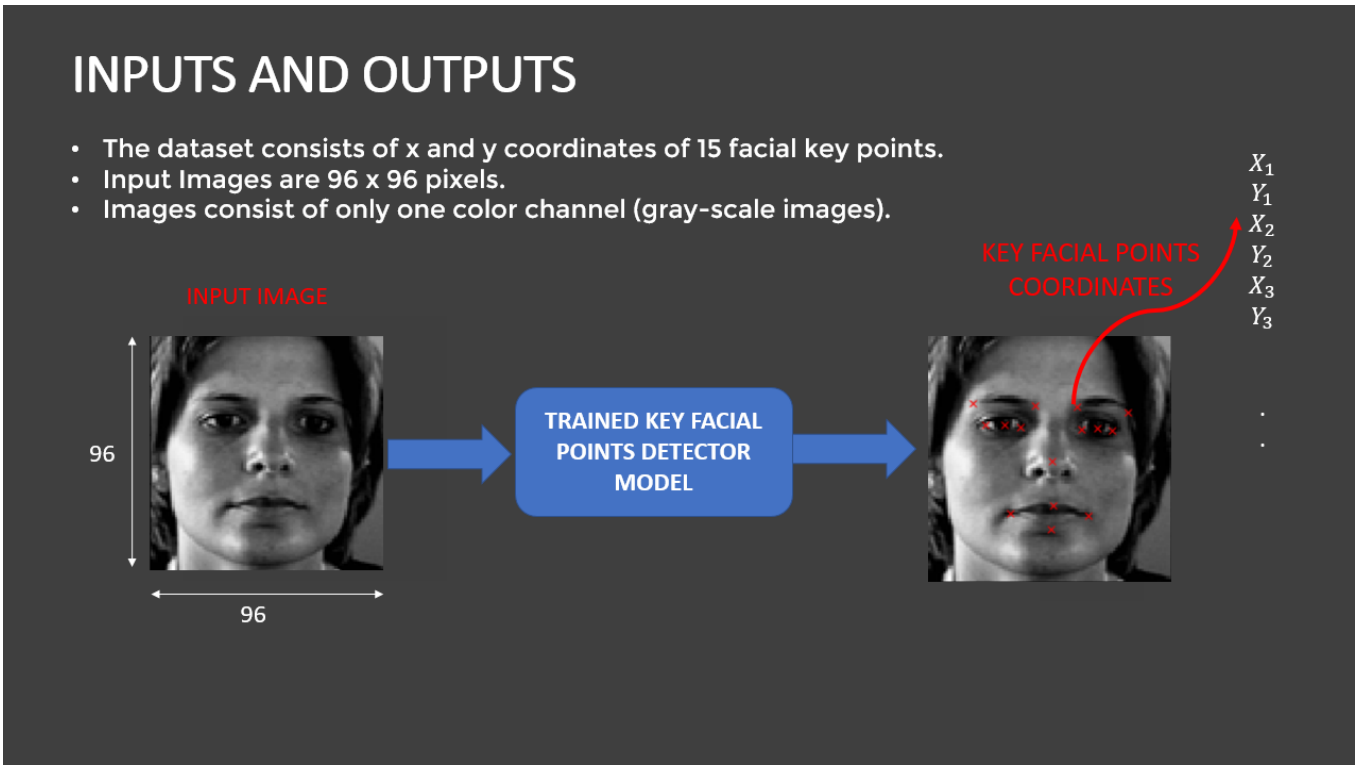
- In this project, we will create a deep learning model based on Convolutional Neural Network and Residual Blocks to predict facial key-points.
- Facial Key-Point Detection serves as a basis for Emotional AI applications like detecting customer emotional responses to Ads and Driver Monitoring Systems.
- :) AFFECTIVA is one of the leading players in Emotional AI and their software detects human emotions, complex cognitive states, and behaviours.
- Check this out: https://go.affectiva.com/auto

Data Source: https://www.kaggle.com/c/facial-keypoints-detection/data

INSTRUCTOR

- Adjunct professor & online instructor
- Passionate about artificial intelligence, machine learning, and electric vehicles
- Taught 90,000+ students globally
- MBA (2018), Ph.D. (2014), M.A.Sc (2011)

Ryan Ahmed, Ph.D.



TASK #2: IMPORT LIBRARIES/DATASETS AND PERFORM PRELIMINARY DATA PROCESSING

In [2]:

```
# Import the necessary packages
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.python.keras import Sequential
from tensorflow.keras import layers, optimizers
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import *
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, ModelCheckpoint,
from IPython.display import display
from tensorflow.keras import backend as K
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras import optimizers
```

In [3]:

```
# load the data
facialpoints_df = pd.read_csv('KeyFacialPoints.csv')
```

In [4]:

```
facialpoints_df
```

Out[4]:

	left_eye_center_x	left_eye_center_y	right_eye_center_x	right_eye_center_y	left_eye_inn
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	
...	...	...	...	...	

	left_eye_center_x	left_eye_center_y	right_eye_center_x	right_eye_center_y	left_eye_inn
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	
2138	64.152180	30.691592	27.000898	40.868082	
2139	66.683755	34.483429	30.784490	38.578939	

2140 rows × 31 columns

In [5]:

```
facialpoints_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
```

In [6]:

```
# Let's take a look at a sample image
facialpoints_df['Image'][1]
```

```
141 137 132 129 125 118 120 124 126 122 120 128 132 133 125 119 122 123
125 123 121 120 118 116 114 117 126 142 156 156 142 120 114 116 100 93 1
04 119 112 93 95 102 98 90 87 77 66 62 48 49 71 91 102 106 96 79 68 65 6
7 72 216 208 200 190 180 178 188 191 191 192 197 187 191 200 193 205 224
225 218 219 220 219 208 198 203 204 197 196 193 191 196 186 161 151 157
150 140 137 132 129 125 125 121 121 124 125 124 124 129 129 121 121 123
125 124 122 122 121 121 120 117 118 125 146 155 146 131 113 113 114 104
91 104 120 112 95 93 106 99 90 96 83 75 76 67 46 46 72 90 106 107 81 61
57 53 58 200 187 181 179 179 188 190 184 180 188 189 182 188 192 189 205
225 225 224 223 220 215 208 205 208 203 197 199 200 197 194 174 154 146
148 146 141 139 135 131 127 124 122 121 121 122 119 119 125 123 119 120
121 120 122 124 122 122 120 119 121 118 126 146 145 130 117 108 115 115
105 95 103 116 112 98 96 106 105 93 96 92 76 77 80 62 45 55 73 83 100 93
60 46 39 49 199 195 194 190 187 183 187 191 191 185 178 180 190 193 195
209 223 225 225 226 219 211 207 210 209 201 198 202 204 198 187 160 148
143 142 145 144 143 136 133 130 125 123 125 125 122 120 124 121 116 120
117 113 115 119 121 120 117 117 118 118 121 127 138 128 119 111 106 110
114 106 97 106 115 109 98 98 106 108 99 100 93 74 72 78 75 52 50 69 71 8
5 96 72 50 35 40 212 202 196 187 183 186 189 192 190 182 173 180 194 194
200 212 222 223 226 226 216 206 206 214 210 198 200 205 206 196 170 151
```

In [7]:

```
# Since values for the image is given as space separated string, we will need to separate
# Then convert this into numpy array using np.fromstring and convert the obtained 1D arra
facialpoints_df['Image'] = facialpoints_df['Image'].apply(lambda x: np.fromstring(x, dtyp
```

In [8]:

```
# Let's obtain the shape of the resized image
facialpoints_df['Image'][1].shape
```

Out[8]:

```
(96, 96)
```

In [9]:

```
# Let's confirm that there are no null values
facialpoints_df.isnull().sum()
```

Out[9]:

```
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
```

MINI CHALLENGE #1

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

In [55]:

```
facialpoints_df.describe()
```

Out[55]:

	left_eye_center_x	left_eye_center_y	right_eye_center_x	right_eye_center_y	left_eye_in
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	

8 rows × 30 columns

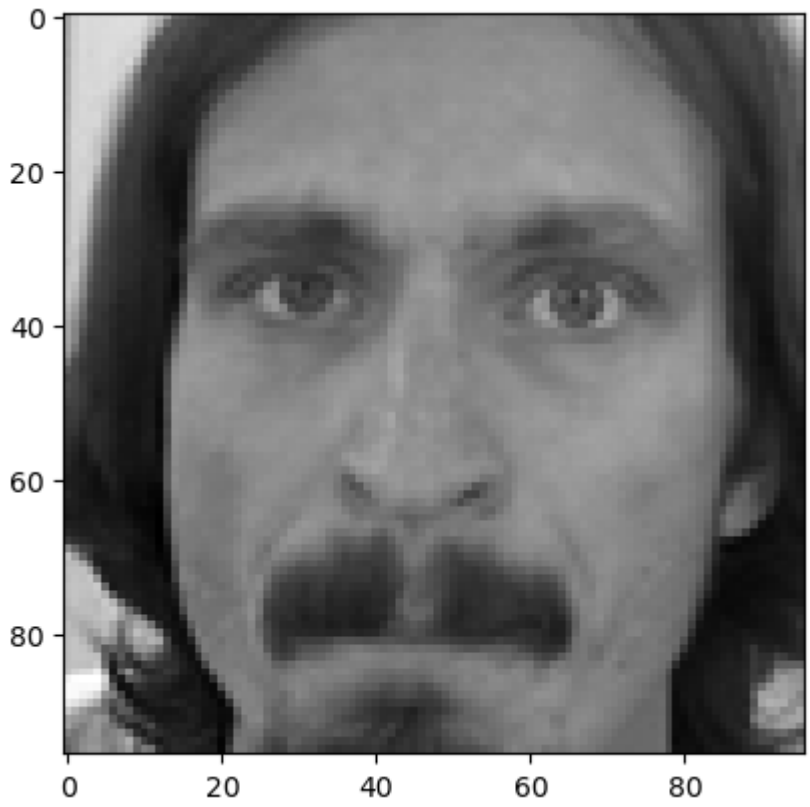
TASK #3: PERFORM IMAGE VISUALIZATION

In [10]:

```
# Plot a random image from the dataset along with facial keypoints.
i = np.random.randint(1, len(facialpoints_df))
plt.imshow(facialpoints_df['Image'][i],cmap='gray')
```

Out[10]:

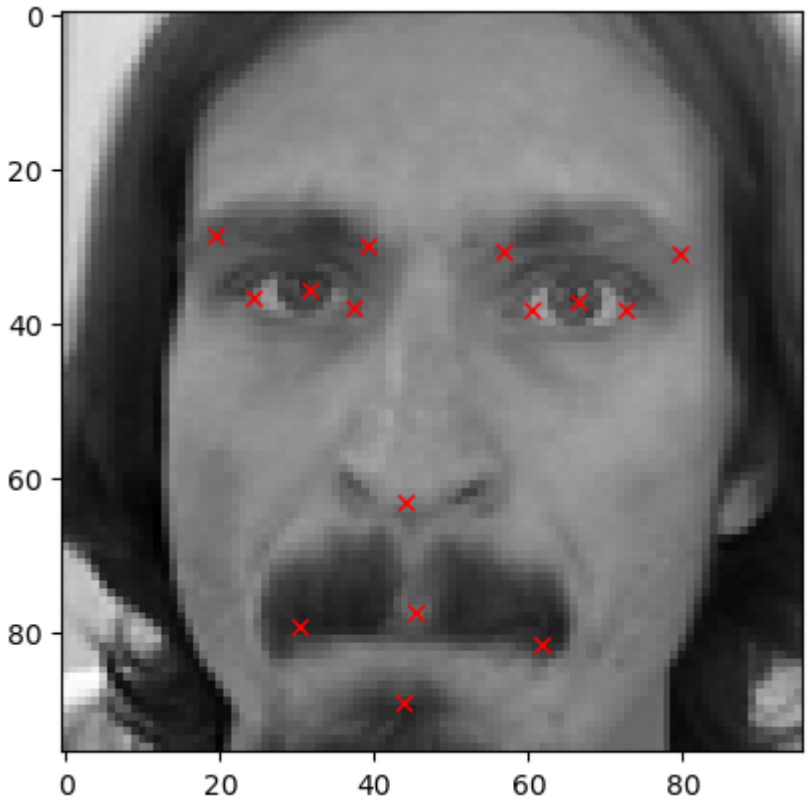
```
<matplotlib.image.AxesImage at 0x1c029bd9280>
```



In [11]:

```
# The (x, y) coordinates for the 15 key features are plotted on top of the image
# Below is a for loop starting from index = 1 to 32 with step of 2
# In the first iteration j would be 1, followed by 3 and so on.
# since x-coordinates are in even columns like 0,2,4,.. and y-coordinates are in odd colu
# we access their value using .loc command, which get the values for coordinates of the i
# in the first iteration df[i][j-1] would be df[i][0] refering the value in 1st column(x-

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



In [57]:

```
import random

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

MINI CHALLENGE #2

- Plot 64 random images from the training data instead of the 16
- HINT: You minght need to choose 'random' to select random images

In [59]:

```
import random

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

for i in range(64):
    ax = fig.add_subplot(8, 8, i + 1)

    img = np.random.randint(1, len(facialpoints_df))


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

TASK #4: PERFORM IMAGE AUGMENTATION

In [13]:

```
# Create a new copy of the dataframe
import copy
facialpoints_df_copy = copy.copy(facialpoints_df)
```

In [14]:

```
# obtain the header of the DataFrame (names of columns)

columns = facialpoints_df_copy.columns[:-1]
columns
```

Out[14]:

```
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')
```

In [15]:

```
# Take a look at the pixel values of a sample image and see if it makes sense!
facialpoints_df['Image'][0]
```

Out[15]:

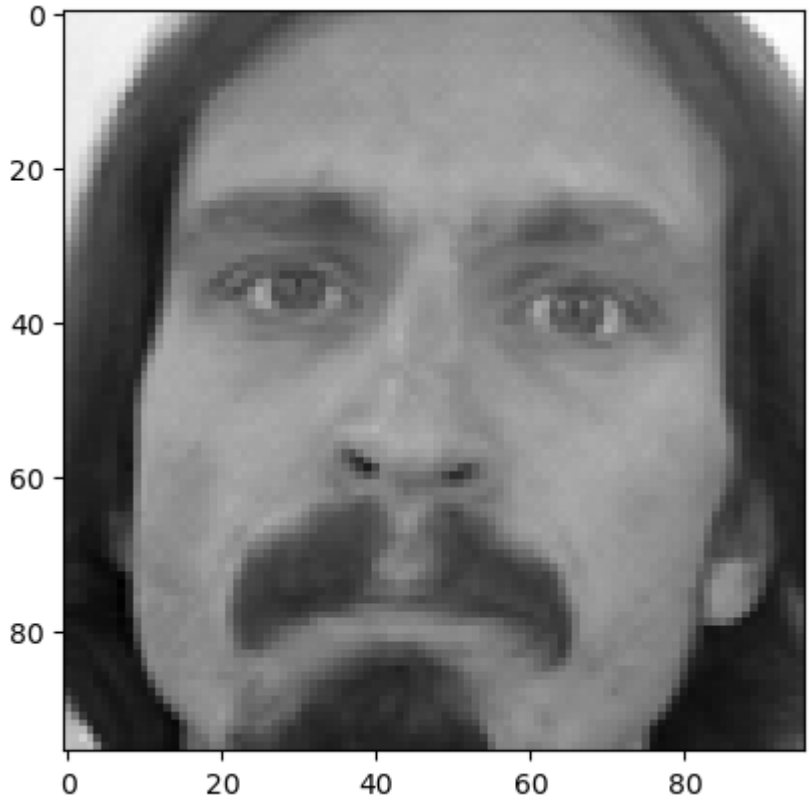
```
array([[238, 236, 237, ..., 250, 250, 250],
       [235, 238, 236, ..., 249, 250, 251],
       [237, 236, 237, ..., 251, 251, 250],
       ...,
       [186, 183, 181, ...,  52,  57,  60],
       [189, 188, 207, ...,  61,  69,  78],
       [191, 184, 184, ...,  70,  75,  90]])
```

In [16]:

```
# plot the sample image
plt.imshow(facialpoints_df['Image'][0], cmap = 'gray')
```

Out[16]:

```
<matplotlib.image.AxesImage at 0x1c02a5a6ee0>
```



In [17]:

```
# Now Let's flip the image column horizontally
facialpoints_df_copy['Image'] = facialpoints_df_copy['Image'].apply(lambda x: np.flip(x,
```

In [18]:

```
# Now take a look at the flipped image and do a sanity check!
# Notice that the values of pixels are now flipped
facialpoints_df_copy['Image'][0]
```

Out[18]:

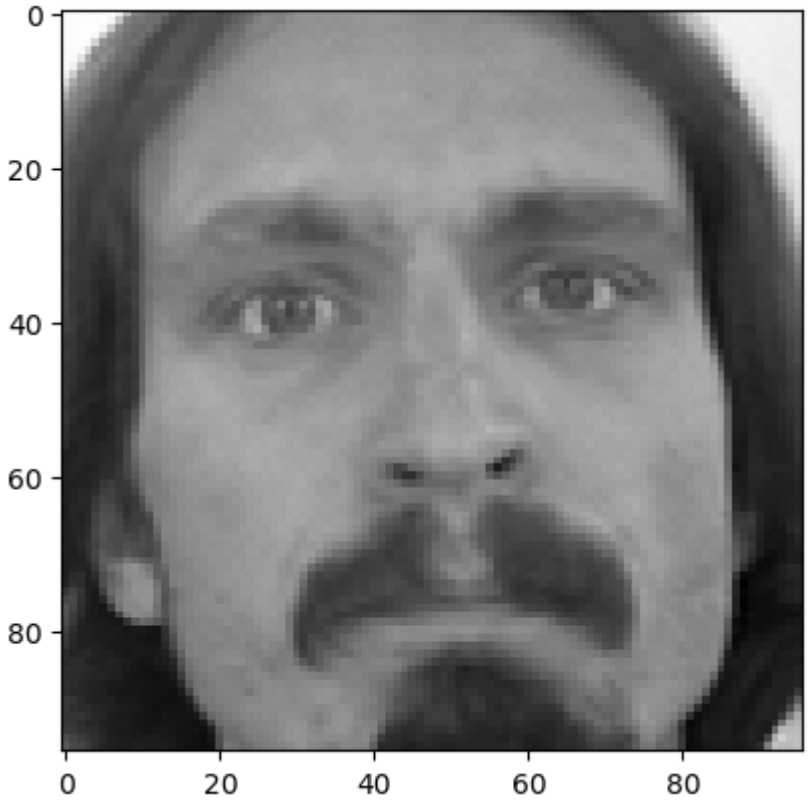
```
array([[250, 250, 250, ..., 237, 236, 238],
       [251, 250, 249, ..., 236, 238, 235],
       [250, 251, 251, ..., 237, 236, 237],
       ...,
       [ 60,  57,  52, ..., 181, 183, 186],
       [ 78,  69,  61, ..., 207, 188, 189],
       [ 90,  75,  70, ..., 184, 184, 191]])
```

In [19]:

```
# Notice that the image is flipped now
plt.imshow(facialpoints_df_copy['Image'][0], cmap = 'gray')
```

Out[19]:

```
<matplotlib.image.AxesImage at 0x1c02a63acd0>
```

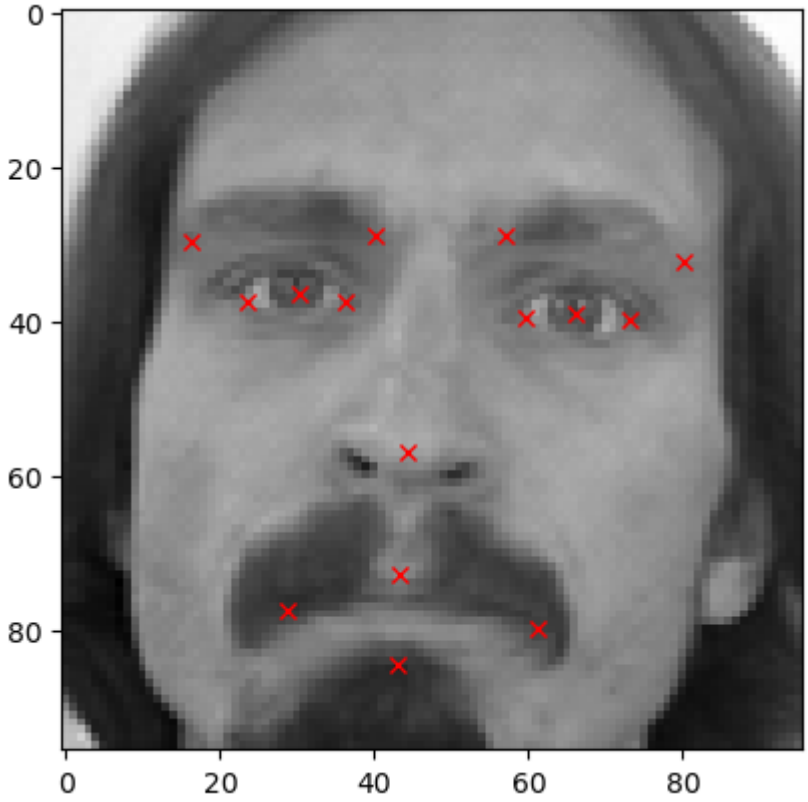


In [20]:

```
# Since we are flipping the images horizontally, y coordinate values would be the same
# X coordinate values only would need to change, all we have to do is to subtract our ini
for i in range(len(columns)):
  if i%2 == 0:
    facialpoints_df_copy[columns[i]] = facialpoints_df_copy[columns[i]].apply(lambda x: 9
```

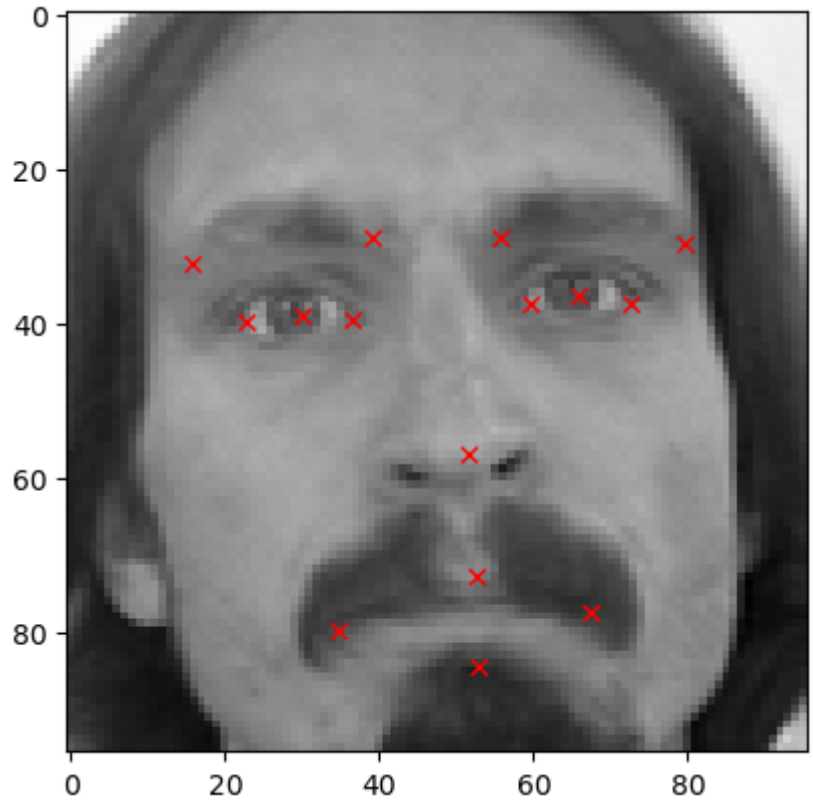
In [21]:

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



In [22]:

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



In [23]:

```
# Concatenate the original dataframe with the augmented dataframe
facialpoints_df_augmented = np.concatenate((facialpoints_df,facialpoints_df_copy))
```

In [24]:

```
facialpoints_df_augmented.shape
```

Out[24]:

```
(4280, 31)
```

In [25]:

```
# Let's try to perform another image augmentation by randomly increasing images brightnes
# We multiply pixel values by random values between 1 and 2 to increase the brightness of
# we clip the value between 0 and 255

import random

facialpoints_df_copy = copy.copy(facialpoints_df)
facialpoints_df_copy['Image'] = facialpoints_df['Image'].apply(lambda x:np.clip(random.ur
facialpoints_df_augmented = np.concatenate((facialpoints_df_augmented, facialpoints_df_cc
facialpoints_df_augmented.shape
```

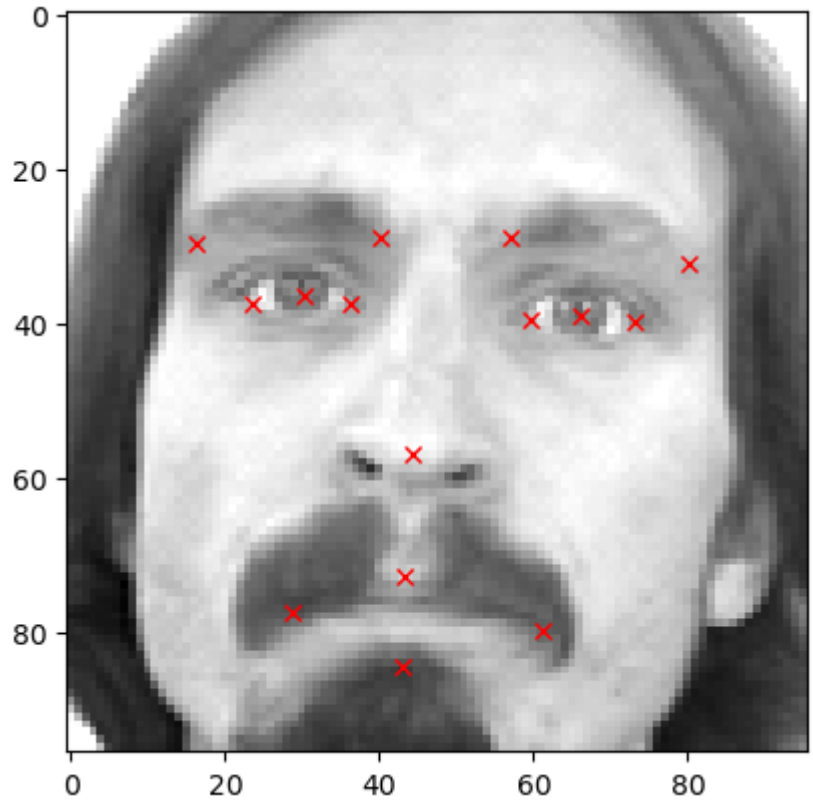
Out[25]:

```
(6420, 31)
```

In [26]:

```
# Let's view image with increased brightness

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



MINI CHALLENGE #3

- Perform image augmentation by decreasing image brightness
- Perform a sanity check and visualize sample images

In [27]:

```
# Randomly decrease image brightness
# Multiply pixel values by random values between 0 and 1 to decrease the brightness of th
# Clip the value between 0 and 255

facialpoints_df_copy = copy.copy(facialpoints_df)
facialpoints_df_copy['Image'] = facialpoints_df['Image'].apply(lambda x:np.clip(random.ur
# facialpoints_df_augmented = np.concatenate((facialpoints_df_augmented, facialpoints_df_
# facialpoints_df_augmented.shape
```

In [28]:

```
facialpoints_df['Image'][0]
```

Out[28]:

```
array([[238, 236, 237, ..., 250, 250, 250],
       [235, 238, 236, ..., 249, 250, 251],
       [237, 236, 237, ..., 251, 251, 250],
       ...,
       [186, 183, 181, ...,  52,  57,  60],
       [189, 188, 207, ...,  61,  69,  78],
       [191, 184, 184, ...,  70,  75,  90]])
```

In [29]:

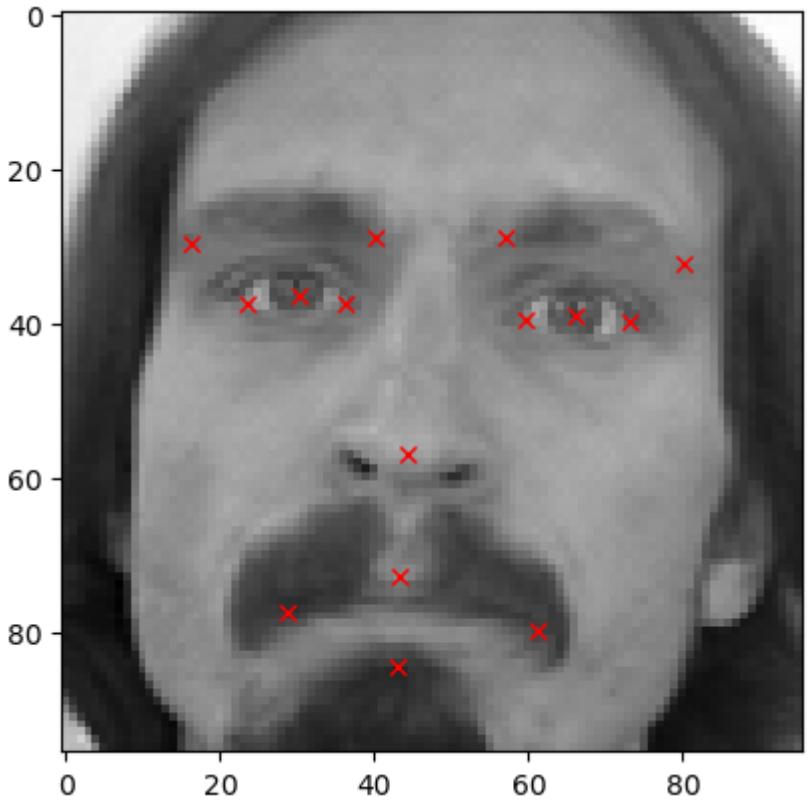
```
facialpoints_df_copy['Image'][0]
```

Out[29]:

```
array([[30.97241284, 30.71214046, 30.84227665, ..., 32.5340471 ,
        32.5340471 , 32.5340471 ],
       [30.58200427, 30.97241284, 30.71214046, ..., 32.40391091,
        32.5340471 , 32.66418329],
       [30.84227665, 30.71214046, 30.84227665, ..., 32.66418329,
        32.66418329, 32.5340471 ],
       ...,
       [24.20533104, 23.81492248, 23.5546501 , ...,  6.7670818 ,
         7.41776274,  7.8081713 ],
       [24.59573961, 24.46560342, 26.938191  , ...,  7.93830749,
         8.979397  , 10.1506227 ],
       [24.85601198, 23.94505867, 23.94505867, ...,  9.10953319,
         9.76021413, 11.71225696]])
```

In [30]:

```
# Let's view a sample image with decreased brightness decreased image
plt.imshow(facialpoints_df_copy['Image'][0], cmap = 'gray')
for j in range(1,31,2):
        plt.plot(facialpoints_df_copy.loc[0][j-1], facialpoints_df_copy.loc[0][j], 'rx')
```



MINI CHALLENGE #4

- 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)

In [31]:

```
facialpoints_df_copy = copy.copy(facialpoints_df)
```

In [32]:

```
# Flip the image column vertically (note that axis = 0)
facialpoints_df_copy['Image'] = facialpoints_df_copy['Image'].apply(lambda x: np.flip(x,
```

In [33]:

```
facialpoints_df['Image'][0]
```

Out[33]:

```
array([[238, 236, 237, ..., 250, 250, 250],
       [235, 238, 236, ..., 249, 250, 251],
       [237, 236, 237, ..., 251, 251, 250],
       ...,
       [186, 183, 181, ...,  52,  57,  60],
       [189, 188, 207, ...,  61,  69,  78],
       [191, 184, 184, ...,  70,  75,  90]])
```

In [34]:

```
facialpoints_df_copy['Image'][0]
```

Out[34]:

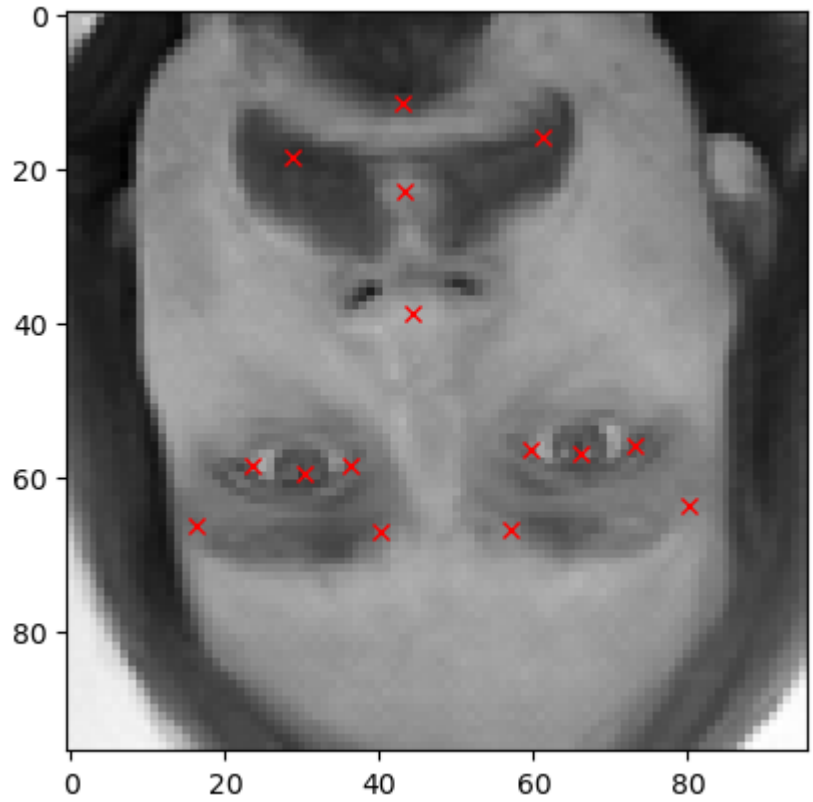
```
array([[191, 184, 184, ...,  70,  75,  90],
       [189, 188, 207, ...,  61,  69,  78],
       [186, 183, 181, ...,  52,  57,  60],
       ...,
       [237, 236, 237, ..., 251, 251, 250],
       [235, 238, 236, ..., 249, 250, 251],
       [238, 236, 237, ..., 250, 250, 250]])
```

In [35]:

```
# Since we are flipping the images vertically, x coordinate values would be the same
# y coordinate values only would need to change, all we have to do is to subtract our ini
for i in range(len(columns)):
  if i%2 == 1:
    facialpoints_df_copy[columns[i]] = facialpoints_df_copy[columns[i]].apply(lambda x: 9
```

In [36]:

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



TASK #5: PERFORM NORMALIZATION AND TRAINING DATA PREPARATION

In [37]:

```
# Obtain the value of 'Images' and normalize it
# Note that 'Images' are in the 31st column but since indexing start from 0, we refer 31s
img = facialpoints_df_augmented[:, 30]
img = img/255.

# Create an empty array of shape (10700, 96, 96, 1) to train the model
X = np.empty((len(img), 96, 96, 1))

# Iterate through the normalized images list and add image values to the empty array
# Note that we need to expand it's dimension from (96,96) to (96,96,1)
for i in range(len(img)):
  X[i,] = np.expand_dims(img[i], axis = 2)

# Convert the array type to float32
X = np.asarray(X).astype(np.float32)
X.shape
```

Out[37]:

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

In [38]:

```
# Obtain the values of key face points coordinates, which are to used as target.
y = facialpoints_df_augmented[:,:30]
y = np.asarray(y).astype(np.float32)
y.shape
```

Out[38]:

```
(6420, 30)
```

In [39]:

```
# Split the data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
```

MINI CHALLENGE #5

- Try a different value for 'test_size'
- Randomly visualize 64 images to make sure that data makes sense prior to training

In [40]:

```
X_train.shape
```

Out[40]:

```
(5778, 96, 96, 1)
```

In [41]:

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

for i in range(64):
    ax = fig.add_subplot(8, 8, i + 1)
    image = plt.imshow(X_train[i].reshape(96,96), cmap = 'gray')
    for j in range(1,31,2):
        plt.plot(y_train[i][j-1], y_train[i][j], 'rx')
```

TASK #6: UNDERSTAND THE THEORY AND INTUITION BEHIND DEEP NEURAL NETWORKS

DEEP LEARNING HISTORY

- There are many trained off the shelve convolutional neural networks that are readily available such as:
  - LeNet-5 (1998): 7 level convolutional neural network developed by LeCun that works in classifying hand writing numbers.
  - AlexNet (2012): Offered massive improvement, error reduction from 26% to 15.3%
  - ZFNEt (2013): achieved error of 14.8%
  - Googlenet/Inception (2014): error reduction to 6.67% which is at par with human level accuracy.
  - VGGNet (2014)
  - ResNet (2015): Residual Neural Network includes "skip connection" feature and therefore enabled training of 152 layers without vanishing gradient issues. Error of 3.57% which is superior than humans.

Source: https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

CONVOLUTIONAL NEURAL NETWORKS

- CNN in action: https://www.cs.ryerson.ca/~aharley/vis/conv/flat.html

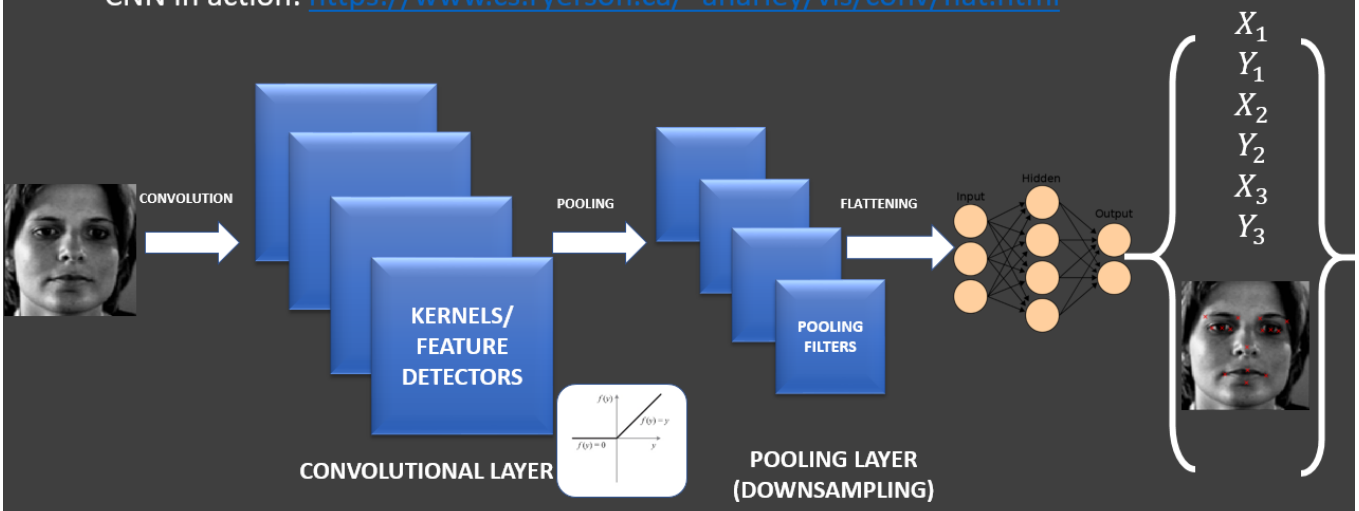
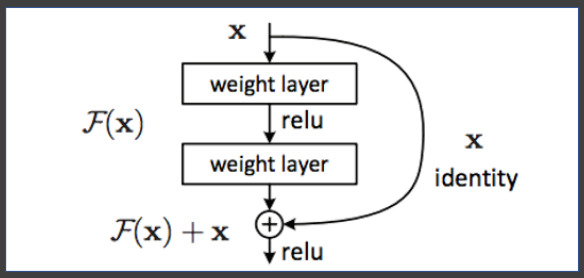


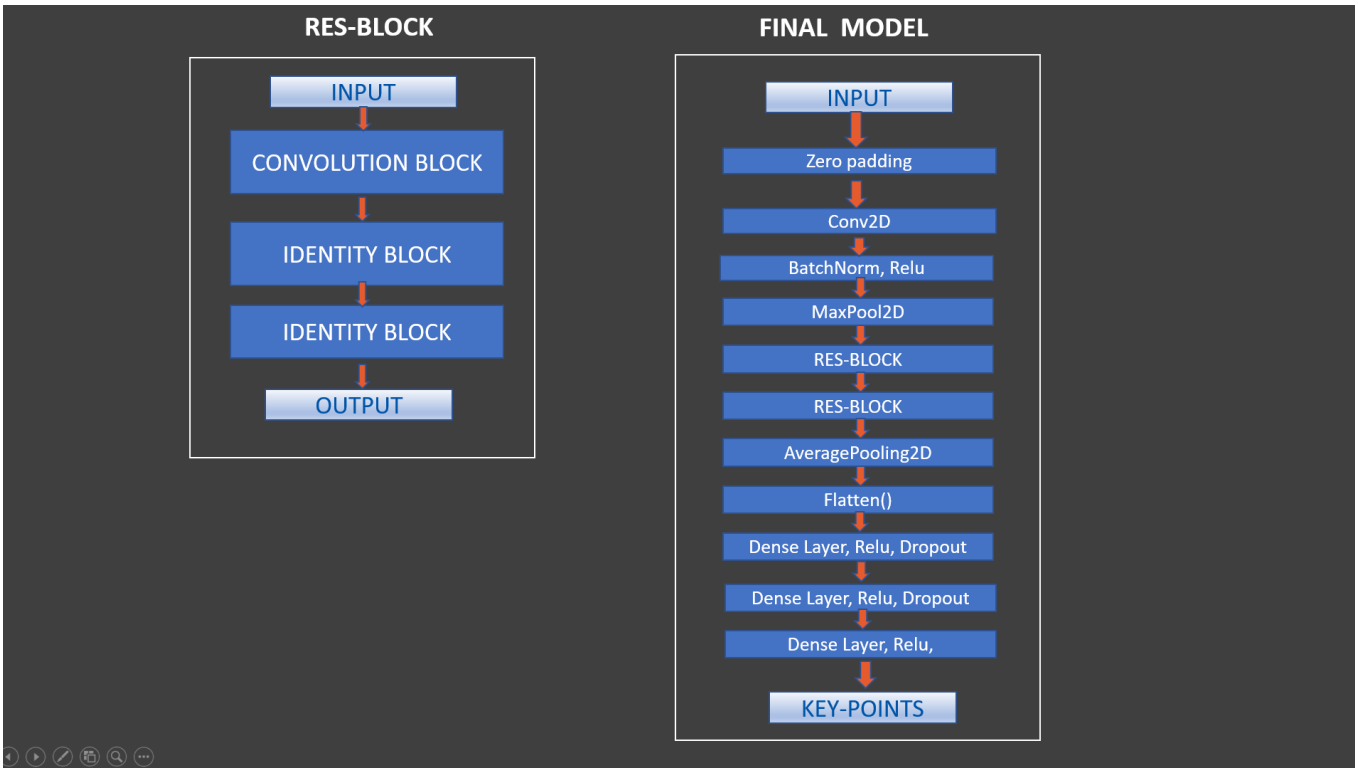
Photo Credit: https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg

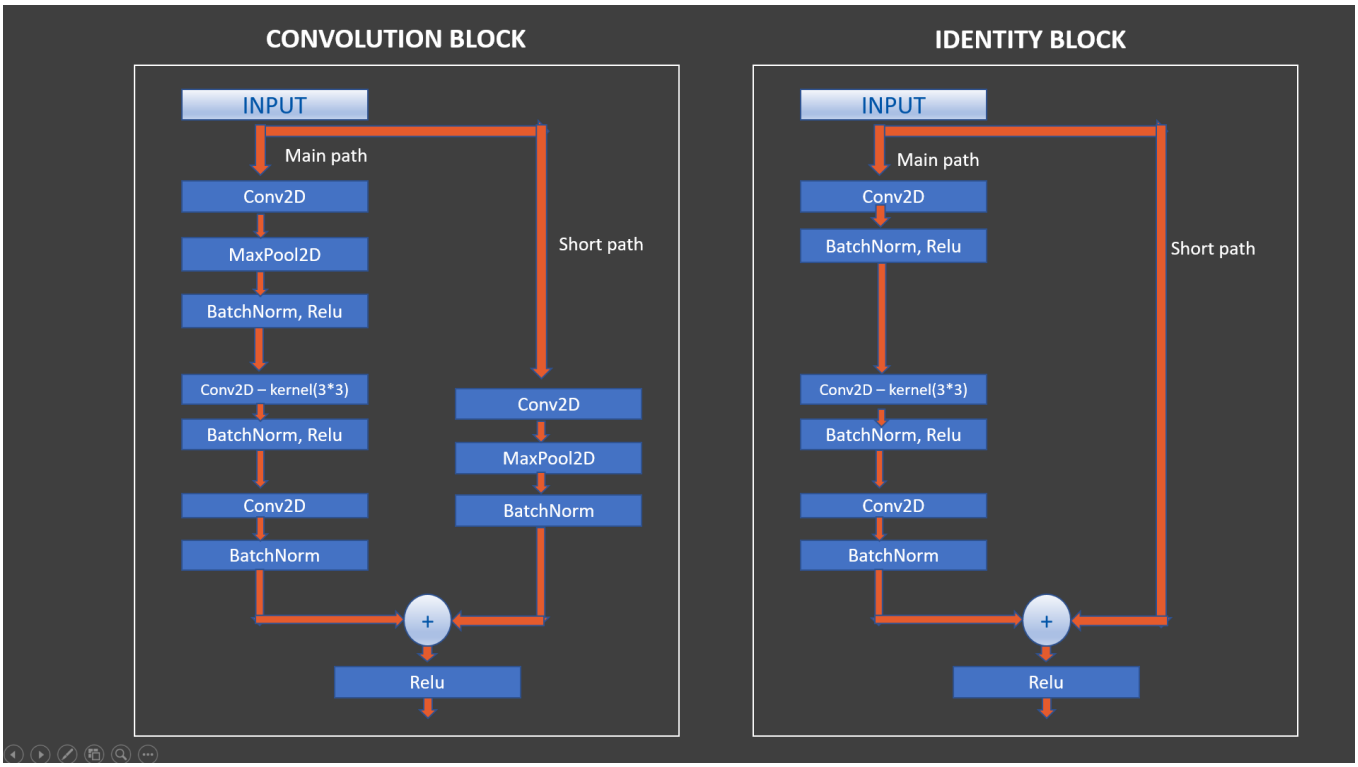
RESNET (RESIDUAL NETWORK)

- 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 the CNN.
- ImageNet contains 11 million images and 11,000 categories.
- ImageNet is used to train ResNet deep network.



TASK #7: BUILD DEEP RESIDUAL NEURAL NETWORK MODEL




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

In [42]:

```
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_initia
  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_'+str(
  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', k
  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)+'_conv_
  X_copy = MaxPool2D((2,2))(X_copy)
  X_copy = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_conv_copy')(X_copy)

  # Add data from main and short paths
  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_i
  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_'+str(
  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_1
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_1_c')(X)

  # Add both paths together (Note that we feed the original input as is hence the name "i
  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_i
  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_'+str(
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_2
X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_2_c')(X)

# Add both paths together (Note that we feed the original input as is hence the name "i
X = Add()([X,X_copy])
X = Activation('relu')(X)

return X
```

In [43]:

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

# Input tensor shape
X_input = Input(input_shape)

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

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

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

# Stage #3
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 = Model( inputs= X_input, outputs = X)
model.summary()
```

```
_3_conv_copy[0][0]']

 activation_12 (Activation)  (None, 5, 5, 512)            0         ['ad
d_3[0][0]']

 res_3_identity_1_a (Conv2D  (None, 5, 5, 128)            65664     ['ac
tivation_12[0][0]']
 )

 bn_3_identity_1_a (BatchNo  (None, 5, 5, 128)            512       ['re
s_3_identity_1_a[0][0]']
 rmalization)

 activation_13 (Activation)  (None, 5, 5, 128)            0         ['bn
_3_identity_1_a[0][0]']

 res_3_identity_1_b (Conv2D  (None, 5, 5, 128)            147584    ['ac
tivation_13[0][0]']
 )
```

MINI CHALLENGE:

- 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.

TASK #8: COMPILE AND TRAIN DEEP LEARNING MODEL

In [44]:

```
adam = tf.keras.optimizers.Adam(lr = 0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)
model.compile(loss="mean_squared_error", optimizer = adam, metrics = ['accuracy'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_r
ate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

In [45]:

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

In [46]:

```
history = model.fit(X_train, y_train, batch_size = 256, epochs= 100, validation_split = 0
```

```
cy: 0.7599
Epoch 76: val_loss did not improve from 39.16723
22/22 [==============================] - 102s 5s/step - loss: 6.0752 - a
ccuracy: 0.7599 - val_loss: 40.4597 - val_accuracy: 0.7197
Epoch 77/100
22/22 [==============================] - ETA: 0s - loss: 6.1332 - accura
cy: 0.7593
Epoch 77: val_loss did not improve from 39.16723
22/22 [==============================] - 102s 5s/step - loss: 6.1332 - a
ccuracy: 0.7593 - val_loss: 39.7377 - val_accuracy: 0.7059
Epoch 78/100
22/22 [==============================] - ETA: 0s - loss: 5.9625 - accura
cy: 0.7584
Epoch 78: val_loss improved from 39.16723 to 37.37673, saving model to w
eights.hdf5
22/22 [==============================] - 105s 5s/step - loss: 5.9625 - a
ccuracy: 0.7584 - val_loss: 37.3767 - val_accuracy: 0.7197
Epoch 79/100
22/22 [==============================] - ETA: 0s - loss: 5.6913 - accura
cy: 0.7626
```

MINI CHALLENGE:

- Experiment with changing the batch size and validation split value
- Comment on your answer

TASK #9: ASSESS TRAINED MODEL PERFORMANCE

In [47]:

```
# Evaluate trained model

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

```
21/21 [==============================] - 3s 152ms/step - loss: 46.8234 - a
ccuracy: 0.7150
Accuracy : 0.7149532437324524
```

In [48]:

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

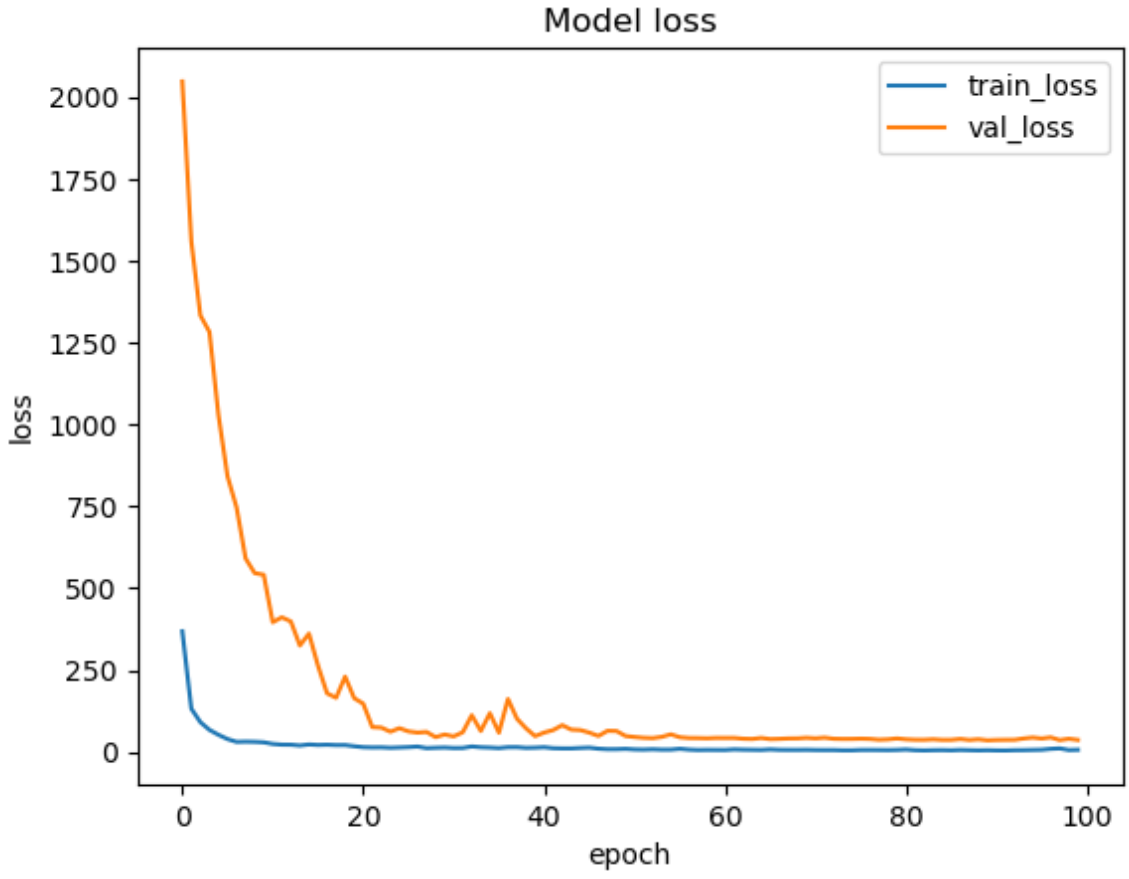
Out[48]:

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

In [49]:

```
# 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()
```



In [50]:

```
# Make prediction using the testing dataset
df_predict = model.predict(X_test)
```

```
21/21 [==============================] - 4s 147ms/step
```

In [51]:

```
# Print the rmse loss values

from sklearn.metrics import mean_squared_error
from math import sqrt

rms = sqrt(mean_squared_error(y_test, df_predict))
print("RMSE value : {}".format(rms))
```

```
RMSE value : 6.842761103282911
```

In [52]:

```
# Convert the predicted values into a dataframe

df_predict= pd.DataFrame(df_predict, columns = columns)
df_predict.head()
```

Out[52]:

	left_eye_center_x	left_eye_center_y	right_eye_center_x	right_eye_center_y	left_eye_inner_
0	66.546814	34.878189	28.750193	34.561085	5
1	29.482170	34.187202	59.123112	35.618015	3
2	66.376038	36.496006	31.114544	34.996719	5
3	31.488163	34.503323	62.687824	35.487240	3
4	65.350502	37.453686	29.051720	37.763889	5

5 rows × 30 columns

In [53]:

```
# Plot the test images and their predicted keypoints

fig = plt.figure(figsize=(20, 20))

for i in range(8):
    ax = fig.add_subplot(4, 2, i + 1)
    # Using squeeze to convert the image shape from (96,96,1) to (96,96)
    plt.imshow(X_test[i].squeeze(),cmap='gray')
    for j in range(1,31,2):
            plt.plot(df_predict.loc[i][j-1], df_predict.loc[i][j], 'rx')
```

CONGRATULATIONS ON FINISHING THE PROJECT