TASK #1: PROJECT OVERVIEW

KEY FACIAL POINTS DETECTION

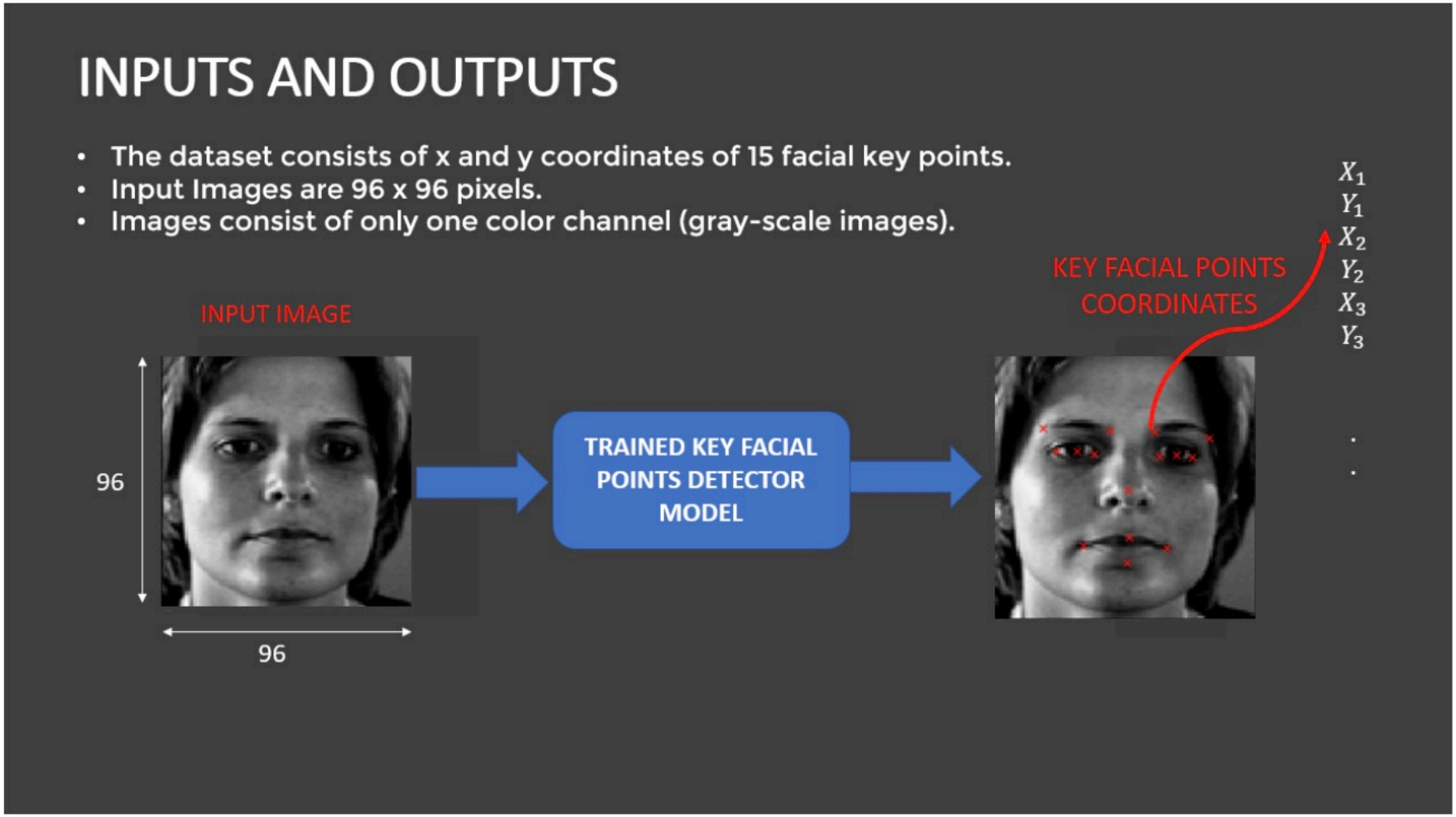
- In this project, we will create a deep learning model based on Convolutional Neural Network and Residual Blocks to predict facial key-points.
- Facial Key-Point Detection serves as a basis for Emotional AI applications like detecting customer emotional responses to Ads and Driver Monitoring Systems.
- :) **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 [10]:
```
pip install tensorflow
```

In [11]:
```
# 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, LearningRateScheduler
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 [12]:
```
# load the data
facialpoints_df = pd.read_csv('KeyFacialPoints.csv')
```

In [13]:
```
facialpoints_df
```

Out[13]:

	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_e
0	66.033564	39.002274	30.227008	36.421678	59.582075	39.647423	73.130346	
1	64.332936	34.970077	29.949277	33.448715	58.856170	35.274349	70.722723	
2	65.057053	34.909642	30.903789	34.909642	59.412000	36.320968	70.984421	
3	65.225739	37.261774	32.023096	37.261774	60.003339	39.127179	72.314713	
4	66.725301	39.621261	32.244810	38.042032	58.565890	39.621261	72.515926	
...	...	...	...	...	...	...	...	
2135	67.180378	35.816373	33.239956	34.921932	59.347973	37.000904	72.667896	

2140 rows × 31 columns

In [14]:
```
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 [15]:
```
# Let's take a look at a sample image
facialpoints_df['Image'][1]
```

```
99 82 73 59 46 39 38 40 46 52 54 48 46 49 53 55 53 55 55 54 52 52 72 94 101 101 99 81 57 34 15 16 20 26 34 41 45
46 51 58 69 89 96 131 130 119 107 92 78 68 65 72 80 94 161 234 248 248 243 220 219 235 234 219 209 193 172 162 1
53 139 129 124 118 112 108 109 105 102 105 106 111 114 112 112 113 115 124 136 134 132 134 130 129 127 121 113 9
4 79 71 66 63 61 66 71 75 79 86 88 88 91 93 92 90 87 81 76 67 55 48 66 88 105 102 82 59 35 16 10 11 16 24 34 50
60 59 54 57 69 88 156 140 121 107 92 77 67 63 62 67 98 177 240 250 247 244 234 232 233 227 215 206 192 169 149 1
37 126 116 112 107 100 99 98 96 100 106 108 111 114 113 115 125 142 151 153 148 141 139 136 126 123 120 116 100
85 81 80 81 87 90 90 94 97 104 108 107 108 103 101 107 106 96 90 85 80 67 50 65 96 107 90 63 40 16 2 1 6 16 23 3
4 52 66 65 57 61 78 167 155 139 119 95 79 64 57 47 64 112 185 239 248 248 244 240 238 233 227 214 199 179 146 11
7 100 88 72 62 59 56 55 54 56 66 83 102 111 111 109 108 118 148 172 172 160 148 140 135 123 119 118 109 97 87 84
85 89 93 91 91 93 94 96 98 100 99 99 101 101 101 99 100 100 97 92 76 60 84 104 94 68 46 20 3 2 4 16 26 34 40 54
62 62 63 69 167 161 147 135 116 90 61 37 38 68 125 193 242 250 248 246 241 238 232 220 207 187 143 102 92 86 66
40 29 23 19 22 30 33 31 38 65 96 106 108 108 104 132 177 192 173 150 145 133 125 121 118 104 87 80 82 85 89 89 8
0 69 62 64 67 71 72 72 74 80 88 90 100 103 103 100 101 99 73 80 105 99 68 41 19 3 1 2 9 20 30 39 46 55 66 71 71
176 163 147 135 125 111 69 25 38 86 137 208 246 249 249 246 242 238 230 218 196 149 98 80 86 120 131 80 23 4 4 8
20 36 43 30 34 64 92 105 107 97 106 163 205 192 163 143 129 123 119 109 93 79 76 82 84 81 67 46 30 32 36 37 41 4
0 43 47 52 55 68 83 93 95 97 107 104 90 90 104 98 70 37 12 1 1 2 7 12 20 28 37 46 55 71 76 178 171 159 137 119 1
11 75 21 46 106 157 220 247 248 247 246 244 239 228 215 172 110 92 90 136 228 238 135 42 10 1 17 39 66 85 59 35
40 73 93 104 101 96 140 201 198 167 143 121 114 113 99 85 76 75 77 70 56 35 25 27 32 19 9 9 7 10 21 37 51 54 56
79 95 100 103 105 95 99 108 102 72 37 10 1 3 11 19 23 23 29 32 34 43 63 75 180 170 161 148 124 93 52 38 70 112 1
74 228 247 249 247 245 245 242 229 200 144 113 143 139 179 250 248 142 50 26 18 27 36 82 120 89 54 45 59 78 93 9
```

In [16]:
```
# values for the image is given as space separated string, we will need to separate the values using ' ' as separator
convert this into numpy array using np.fromstring and convert the obtained 1D array into 2D array of shape (96,96)
oints_df['Image'] = facialpoints_df['Image'].apply(lambda x: np.fromstring(x, dtype= int, sep = ' ').reshape(96,96))
```

In [17]:
```
# Let's obtain the shape of the resized image
facialpoints_df['Image'][1].shape
```

Out[17]: (96, 96)

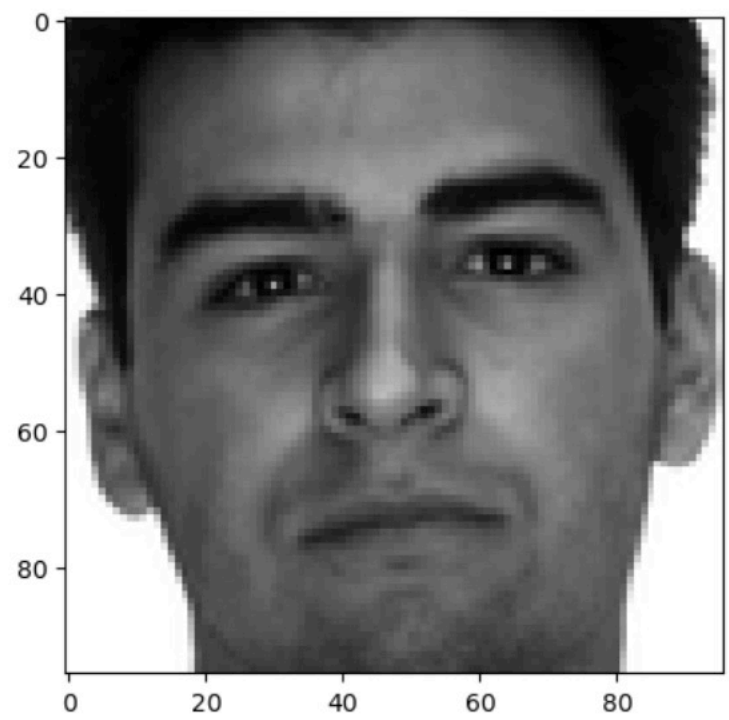
In [18]:
```
# Let's confirm that there are no null values
facialpoints_df.isnull().sum()
```

Out[18]:
```
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
```

TASK #3: PERFORM IMAGE VISUALIZATION

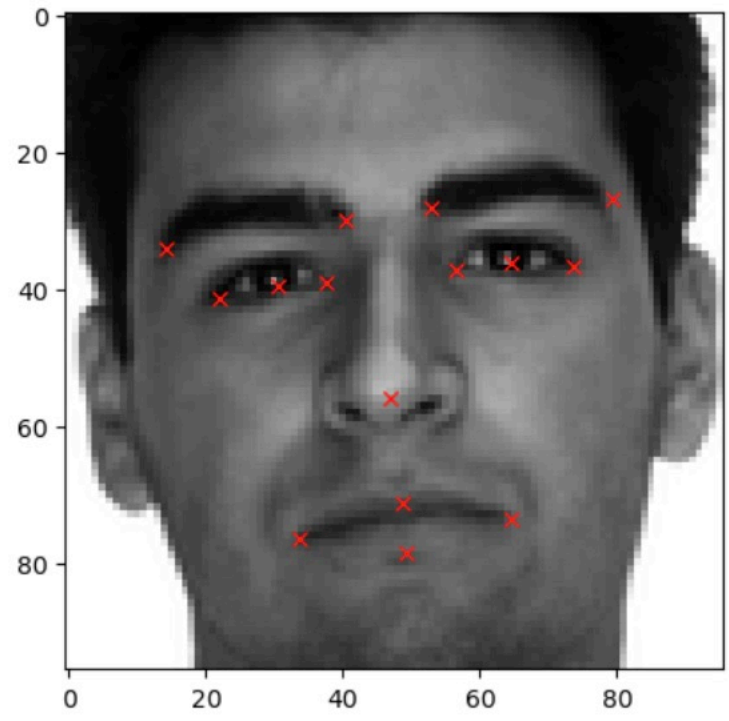
In [19]:
```
# Plot a random image from the dataset along with facial keypoints.
i = np.random.randint(1, len(facialpoints_df))
plt.imshow(facialpoints_df['Image'][i],cmap='gray')
```

Out[19]:
```
<matplotlib.image.AxesImage at 0x3504e93d0>
```



In [20]:
```
# 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 columns like 1,3,5,..
# we access their value using .loc command, which get the values for coordinates of the image based on the column it
# in the first iteration df[i][j-1] would be df[i][0] refering the value in 1st column(x-coordinate) of the image in

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')
```

```
/var/folders/83/bqlfyrvj1319dw3cqtd4cslc0000gn/T/ipykernel_2349/2328267439.py:12: FutureWarning: Series.__getitem__
treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consi
stent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  plt.plot(facialpoints_df.loc[i][j-1], facialpoints_df.loc[i][j], 'rx')
```



In [21]:
```
# 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')
```

```
/var/folders/83/bqlfyrvj1319dw3cqtd4cslc0000gn/T/ipykernel_2349/1638532457.py:8: FutureWarning: Series.__getitem__
treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consi
stent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  plt.plot(facialpoints_df.loc[i][j-1], facialpoints_df.loc[i][j], 'rx')
```

MINI CHALLENGE

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

In [22]:
```
facialpoints_df.describe()
```

Out[22]:

	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_
count	2140.000000	2140.000000	2140.000000	2140.000000	2140.000000	2140.000000	2140.000000	
mean	66.221549	36.842274	29.640269	37.063815	59.272128	37.856014	73.412473	
std	2.087683	2.294027	2.051575	2.234334	2.005631	2.034500	2.701639	
min	47.835757	23.832996	18.922611	24.773072	41.779381	27.190098	52.947144	
25%	65.046300	35.468842	28.472224	35.818377	58.113054	36.607950	71.741978	
50%	66.129065	36.913319	29.655440	37.048085	59.327154	37.845220	73.240045	
75%	67.332093	38.286438	30.858673	38.333884	60.521492	39.195431	74.978684	
max	78.013082	46.132421	42.495172	45.980981	69.023030	47.190316	87.032252	

8 rows × 30 columns

In [23]:
```
maximum_right_eye_x = facialpoints_df['right_eye_center_x'].max()
minimum_right_eye_x = facialpoints_df['right_eye_center_x'].min()
average_right_eye_x = facialpoints_df['right_eye_center_x'].mean()

print(f"average: {average_right_eye_x}\n",
      f"max: {maximum_right_eye_x}\n",
      f"min: {minimum_right_eye_x}")
```

```
average: 29.640268564561495
 max: 42.495171727
 min: 18.9226106286
```

MINI CHALLENGE #2

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

In [28]:
```
# # Let's view more images in a grid format
# fig = plt.figure(figsize=(20, 20))

# rand_indices = sorted(random.sample(range(len(facialpoints_df)), 64))

# for k, i in enumerate(rand_indices):
#     ax = fig.add_subplot(8, 8, k + 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')
```

TASK #4: PERFORM IMAGE AUGMENTATION

In [29]:
```
# Create a new copy of the dataframe
import copy
facialpoints_df_copy = copy.copy(facialpoints_df)
```

In [30]:
```
# obtain the header of the DataFrame (names of columns)

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

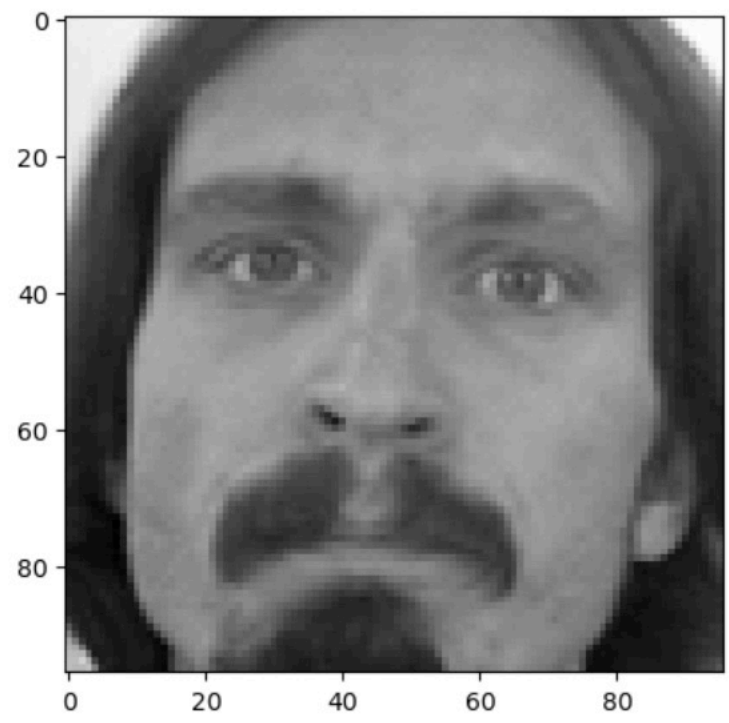
Out[30]:
```
Index(['left_eye_center_x', 'left_eye_center_y', 'right_eye_center_x',
       'right_eye_center_y', 'left_eye_inner_corner_x',
       'left_eye_inner_corner_y', 'left_eye_outer_corner_x',
       'left_eye_outer_corner_y', 'right_eye_inner_corner_x',
       'right_eye_inner_corner_y', 'right_eye_outer_corner_x',
       '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 [31]:
```
# Take a look at the pixel values of a sample image and see if it makes sense!
facialpoints_df['Image'][0]
```

Out[31]:
```
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 [32]:
```
# plot the sample image
plt.imshow(facialpoints_df['Image'][0], cmap = 'gray')
```

Out[32]:
```
<matplotlib.image.AxesImage at 0x37cb26ed0>
```



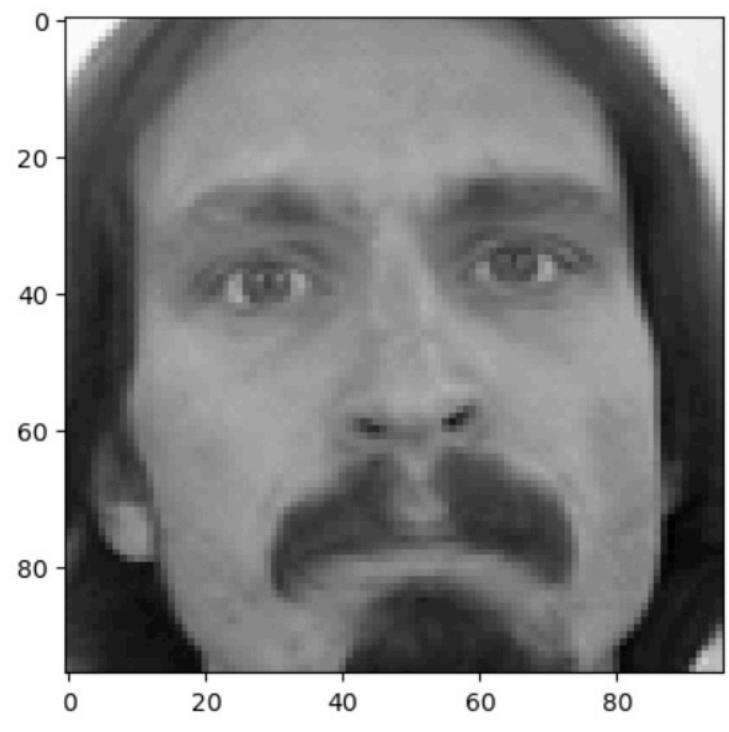
In [33]:
```
# Now Let's flip the image column horizontally
facialpoints_df_copy['Image'] = facialpoints_df_copy['Image'].apply(lambda x: np.flip(x, axis = 1))
```

In [34]:
```
# 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[34]:
```
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 [35]:
```
# Notice that the image is flipped now
plt.imshow(facialpoints_df_copy['Image'][0], cmap = 'gray')
```

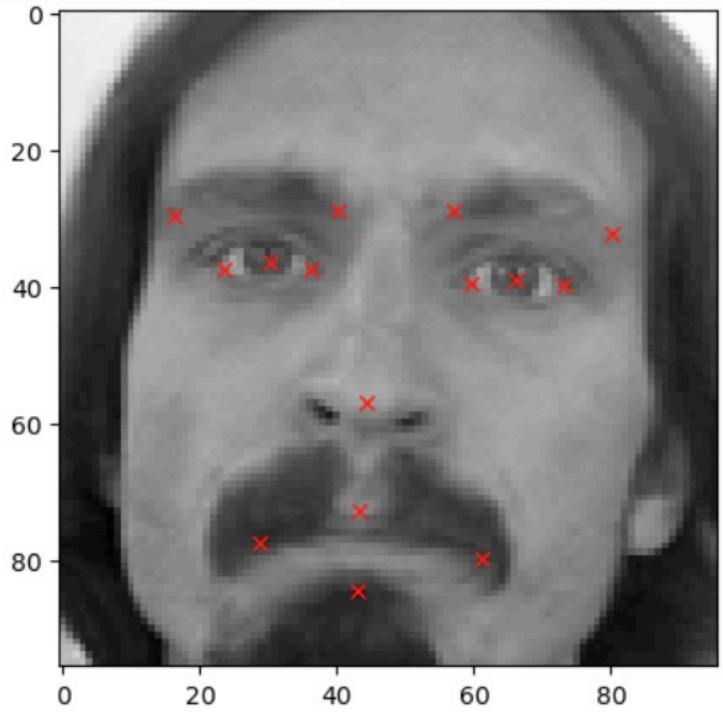
Out[35]:
```
<matplotlib.image.AxesImage at 0x37cbf7410>
```



In [36]:
```
# 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 initial x-coordinate values fr
for i in range(len(columns)):
  if i%2 == 0:
    facialpoints_df_copy[columns[i]] = facialpoints_df_copy[columns[i]].apply(lambda x: 96. - float(x) )
```

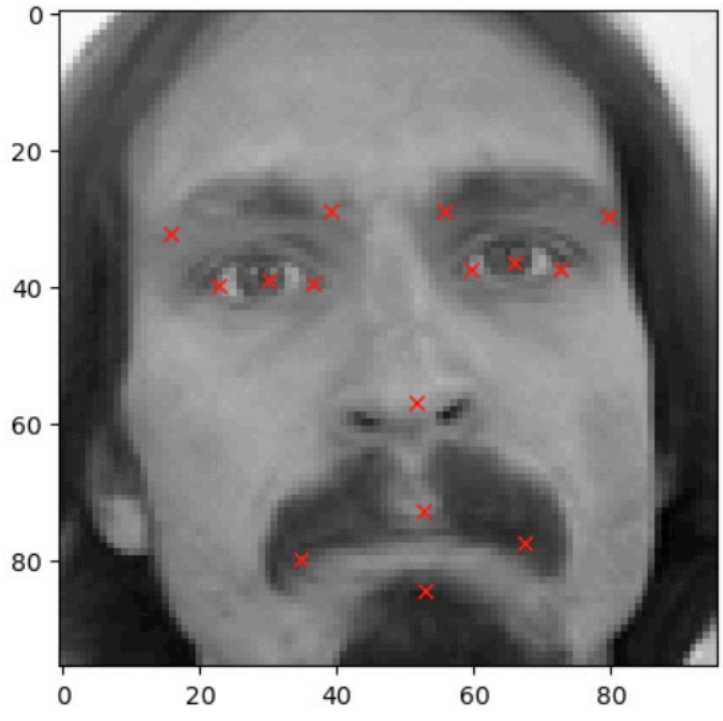
In [37]:
```
# View the Original image
plt.imshow(facialpoints_df['Image'][0],cmap='gray')
for j in range(1, 31, 2):
        plt.plot(facialpoints_df.loc[0][j-1], facialpoints_df.loc[0][j], 'rx')
```

```
/var/folders/83/bqlfyrvj1319dw3cqtd4cslc0000gn/T/ipykernel_2349/3371465537.py:4: FutureWarning: Series.__getitem__
treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consi
stent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  plt.plot(facialpoints_df.loc[0][j-1], facialpoints_df.loc[0][j], 'rx')
```



In [38]:
```
# 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')
```

```
/var/folders/83/bqlfyrvj1319dw3cqtd4cslc0000gn/T/ipykernel_2349/597616358.py:4: FutureWarning: Series.__getitem__ t
reating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consis
tent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  plt.plot(facialpoints_df_copy.loc[0][j-1], facialpoints_df_copy.loc[0][j], 'rx')
```



In [39]:
```
# Concatenate the original dataframe with the augmented dataframe
facialpoints_df_augmented = np.concatenate((facialpoints_df,facialpoints_df_copy))
```

In [40]:
```
facialpoints_df_augmented.shape
```

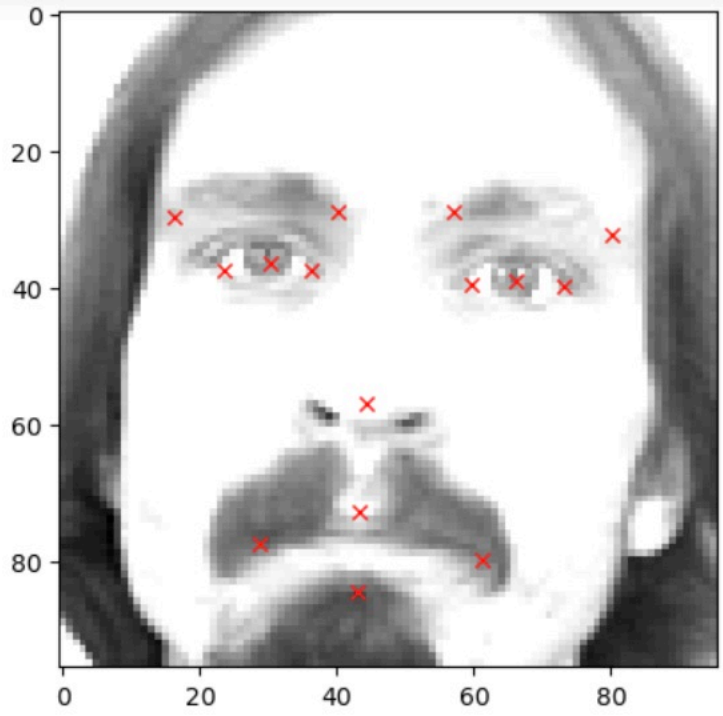
Out[40]: (4280, 31)

In [41]:
```
# Let's try to perform another image augmentation by randomly increasing images brightness
# We multiply pixel values by random values between 1 and 2 to increase the brightness of the image
# 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.uniform(1, 2) * x, 0.0, 255.0
facialpoints_df_augmented = np.concatenate((facialpoints_df_augmented, facialpoints_df_copy))
facialpoints_df_augmented.shape
```

Out[41]: (6420, 31)

In [42]:
```
# 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')
```

```
/var/folders/83/bqlfyrvj1319dw3cqtd4cslc0000gn/T/ipykernel_2349/2235998635.py:5: FutureWarning: Series.__getitem__
treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consi
stent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  plt.plot(facialpoints_df_copy.loc[0][j-1], facialpoints_df_copy.loc[0][j], 'rx')
```



MINI CHALLENGE:

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

In [43]:
```
# Randomly decrease image brightness
# Multiply pixel values by random values between 0 and 1 to decrease the brightness of the image
# 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.uniform(0, 0.2) * x, 0.0, 255
# facialpoints_df_augmented = np.concatenate((facialpoints_df_augmented, facialpoints_df_copy))
# facialpoints_df_augmented.shape
```

In [44]:
```
facialpoints_df['Image'][0]
```

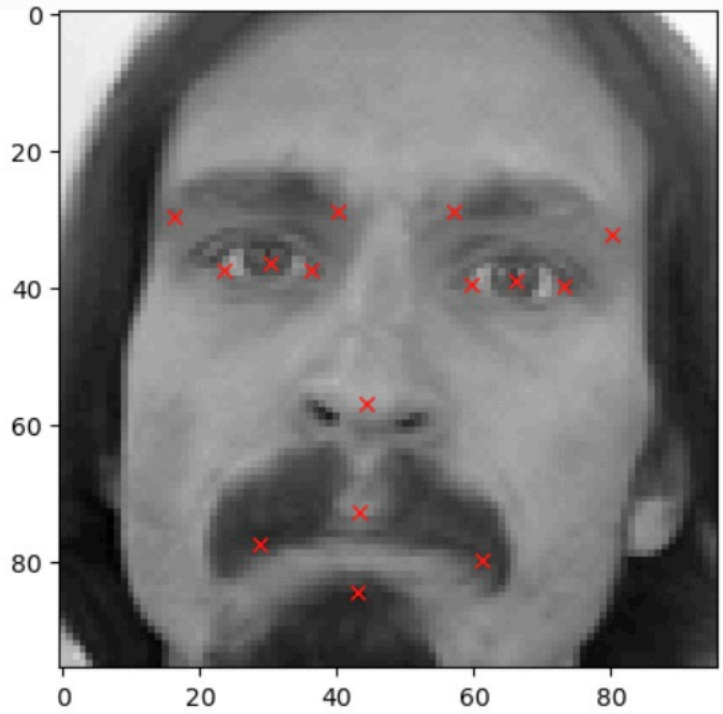
Out[44]:
```
array([[238, 236, 237, ..., 250, 250, 250],
       [235, 238, 236, ..., 249, 250, 251],
       [237, 236, 237, ..., 251, 251, 250],
       ...,
       [186, 183, 181, ...,  52,  57,  60],
       [189, 188, 207, ...,  61,  69,  78],
       [191, 184, 184, ...,  70,  75,  90]])
```

In [45]:
```
facialpoints_df_copy['Image'][0]
```

Out[45]:
```
array([[7.45323357, 7.39060136, 7.42191747, ..., 7.82902686, 7.82902686,
        7.82902686],
       [7.35928525, 7.45323357, 7.39060136, ..., 7.79771076, 7.82902686,
        7.86034297],
       [7.42191747, 7.39060136, 7.42191747, ..., 7.86034297, 7.86034297,
        7.82902686],
       ...,
       [5.82479599, 5.73084766, 5.66821545, ..., 1.62843759, 1.78501813,
        1.87896645],
       [5.91874431, 5.8874282 , 6.48243424, ..., 1.91028255, 2.16081141,
        2.44265638],
       [5.98137652, 5.76216377, 5.76216377, ..., 2.19212752, 2.34870806,
        2.81844967]])
```

In [46]:
```
# 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')
```

```
/var/folders/83/bqlfyrvj1319dw3cqtd4cslc0000gn/T/ipykernel_2349/1913154507.py:4: FutureWarning: Series.__getitem__
treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consi
stent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  plt.plot(facialpoints_df_copy.loc[0][j-1], facialpoints_df_copy.loc[0][j], 'rx')
```



MINI CHALLENGE:

- 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 [47]:
```
facialpoints_df_copy = copy.copy(facialpoints_df)
```

In [48]:
```
# Flip the image column vertically (note that axis = 0)
facialpoints_df_copy['Image'] = facialpoints_df_copy['Image'].apply(lambda x: np.flip(x, axis = 0))
```

In [49]:
```
facialpoints_df['Image'][0]
```

Out[49]:
```
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]])
```

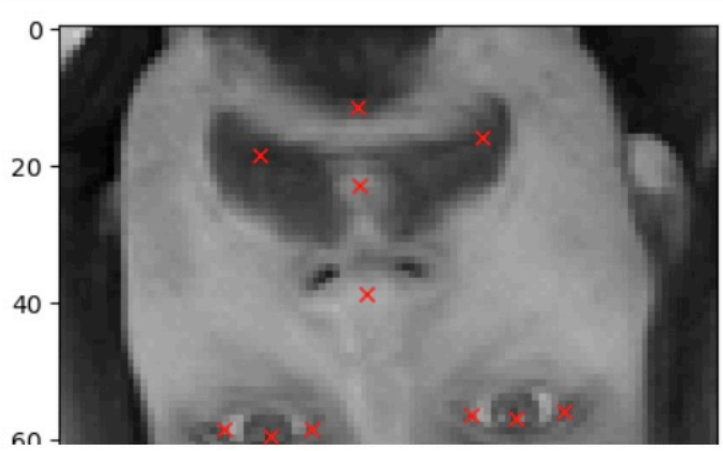
In [50]:
```
facialpoints_df_copy['Image'][0]
```

Out[50]:
```
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 [51]:
```
# 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 initial y-coordinate values fr
for i in range(len(columns)):
  if i%2 == 1:
    facialpoints_df_copy[columns[i]] = facialpoints_df_copy[columns[i]].apply(lambda x: 96. - float(x) )
```

In [52]:
```
# 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')
```

```
/var/folders/83/bqlfyrvj1319dw3cqtd4cslc0000gn/T/ipykernel_2349/597616358.py:4: FutureWarning: Series.__getitem_
_ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels
(consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  plt.plot(facialpoints_df_copy.loc[0][j-1], facialpoints_df_copy.loc[0][j], 'rx')
```



In [53]:
```
# Concatenate the original dataframe with the augmented dataframe
facialpoints_df_augmented = np.concatenate((facialpoints_df_augmented,facialpoints_df_copy))
facialpoints_df_augmented
```

Out[53]:
```
array([[66.0335639098, 39.0022736842, 30.2270075188, ..., 43.1307067669,
        84.4857744361, 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]])],
[64.3329361702, 34.9700765957, 29.9492765957, ..., 45.4679148936,
 85.48017021279999, array([[219, 215, 204, ...,  92,  88,  84],
                           [222, 219, 220, ...,  92,  88,  86],
                           [231, 224, 212, ...,  77,  80,  84],
                           ...,
                           [  1,   1,   1, ...,   1,   1,   1],
                           [  1,   1,   1, ...,   1,   1,   1],
                           [  1,   1,   1, ...,   1,   1,   1]])],
[65.0570526316, 34.9096421053, 30.9037894737, ..., 47.2749473684,
 78.6593684211, array([[144, 142, 159, ..., 208, 207, 207],
                       [143, 142, 161, ..., 208, 208, 207],
                       [143, 140, 160, ..., 209, 209, 207],
                       ...,
                       [ 66,  70,  69, ...,  81, 134, 194],
                       [ 65,  69,  71, ...,  75,  83, 109],
                       [ 65,  68,  70, ...,  78,  78,  77]])],
...,
[68.4308662521, 57.3480248751, 28.8958570552, ..., 47.1767386782,
 6.455477556100007, array([[128, 136, 135, ...,  39,  51,  75],
                           [130, 136, 137, ...,  38,  45,  64],
                           [129, 134, 134, ...,  37,  39,  55],
                           ...,
                           [ 43,  52,  46, ..., 195, 198, 198],
                           [ 35,  47,  46, ..., 197, 199, 197],
                           [ 31,  40,  47, ..., 198, 198, 197]])],
[64.152179592, 65.3084081634, 27.0008979594, ..., 58.6110857142,
 15.644457142799993, array([[ 83,  84,  84, ..., 179, 177,  57],
                            [ 82,  83,  83, ..., 177, 172,  45],
                            [ 82,  83,  83, ..., 179, 164,  36],
                            ...,
                            [  0,   9,   8, ...,   6,  10,  25],
                            [  0,   3,   8, ...,   9,  16,  36],
                            [  7,   1,   5, ...,   9,  20,  38]])],
[66.6837551021, 61.5165714288, 30.7844897957, ...,
 52.92337132270001, 13.338938247399994,
 array([[120, 124, 129, ..., 125, 124, 119],
        [116, 119, 125, ..., 122, 121, 120],
        [116, 118, 123, ..., 118, 116, 117],
        ...,
        [ 91,  26,  16, ...,   5,   8,   5],
        [ 70,  27,  17, ...,   5,   7,   4],
        [ 68,  19,  19, ...,   3,   8,   3]])]], dtype=object)
```

TASK #5: PERFORM NORMALIZATION AND TRAINING DATA PREPARATION

In [54]:
```
# Obtain the value of 'Images' and normalize it
# Note that 'Images' are in the 31st column but since indexing start from 0, we refer 31st column by 30
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[54]:
```
(8560, 96, 96, 1)
```

In [55]:
```
# 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[55]:
```
(8560, 30)
```

In [56]:
```
# Split the data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

MINI CHALLENGE:

- Try a different value for 'test_size' and check the size of the training and testing datasets
- Randomly visualize 64 images to make sure that data makes sense prior to training

In [57]:
```
X_train.shape
```

Out[57]:
```
(6848, 96, 96, 1)
```

In [58]:
```
X_test.shape
```

Out[58]:
```
(1712, 96, 96, 1)
```

In [59]:
```
# Let's view more images in a grid format
fig = plt.figure(figsize=(20, 20))

# Random selection of indices
rand_indices = sorted(random.sample(range(len(X_train)), 64))
print(rand_indices)

for j, i in enumerate(rand_indices):
    ax = fig.add_subplot(8, 8, j + 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')
```

```
[67, 91, 342, 388, 499, 522, 731, 951, 1077, 1239, 1273, 1517, 1522, 1610, 1775, 1845, 1847, 1852, 1888, 1901, 192
1, 2011, 2108, 2112, 2303, 2404, 2453, 2457, 2668, 2722, 2769, 2793, 2807, 2853, 2891, 2901, 3044, 3125, 3217, 352
2, 3579, 3616, 3710, 4135, 4140, 4569, 4629, 4701, 4937, 4940, 4949, 5029, 5150, 5180, 5270, 5374, 5402, 5667, 574
5, 5947, 6041, 6074, 6389, 6791]
```

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

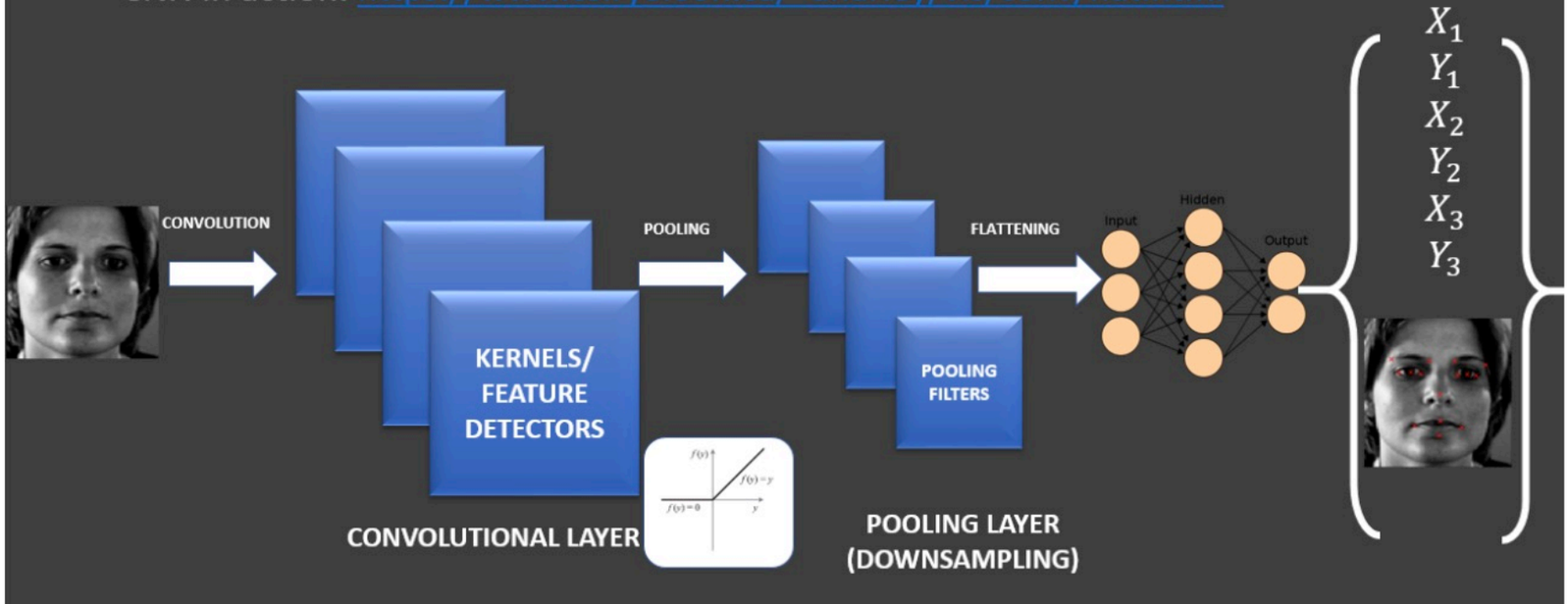
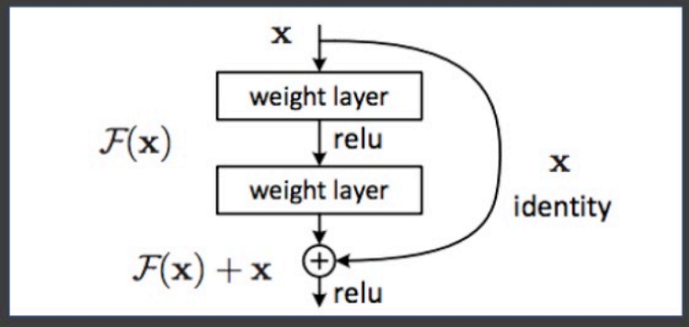


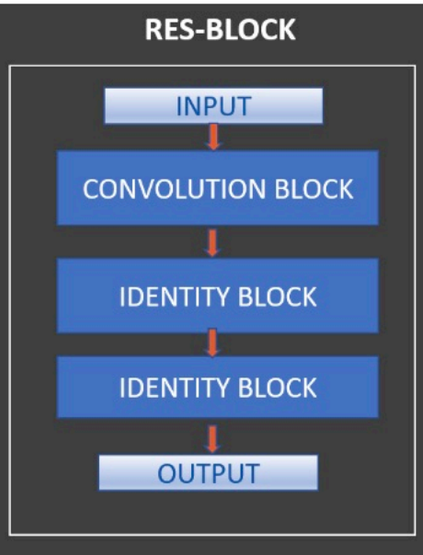
Photo Credit: https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg

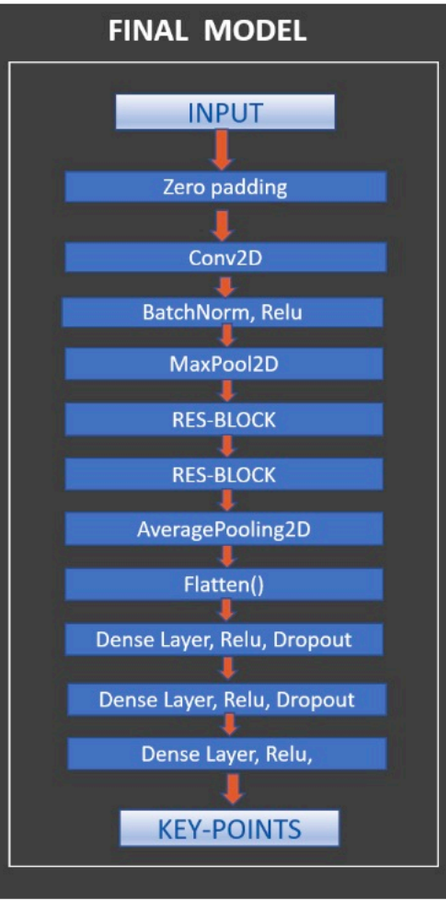
RESNET (RESIDUAL NETWORK)

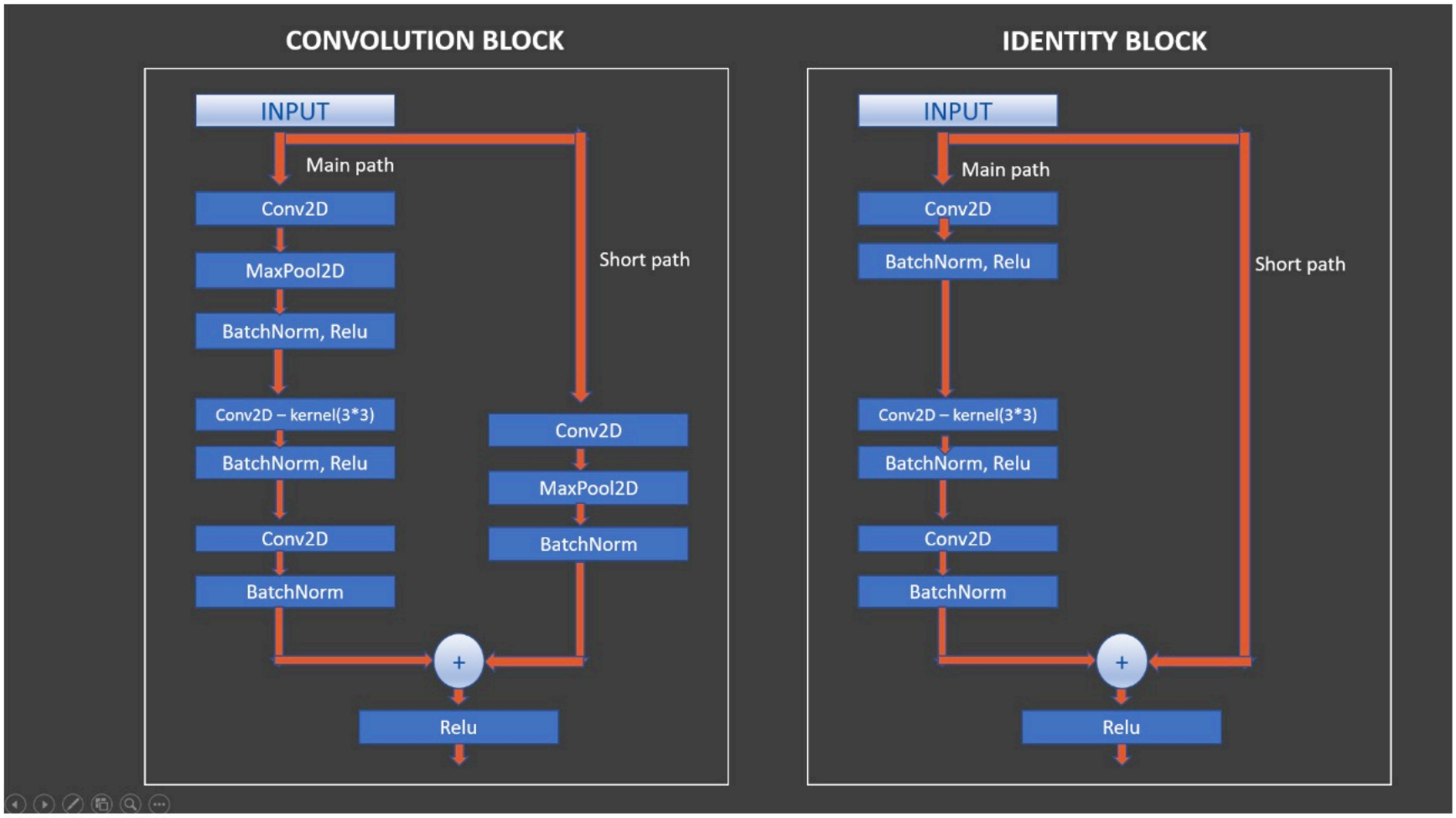
- As CNNs grow deeper, vanishing gradient tend to occur which negatively impact network performance.
- Vanishing gradient problem occurs when the gradient is back-propagated to earlier layers which results in a very small gradient.
- Residual Neural Network includes “skip connection” feature which enables 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







In [60]:

```
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_initializer= glorot_uniform(seed
  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(stage)+'_conv_b', kernel_in
  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', kernel_initializer= glorot_u
  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_copy', kernel_initializer=
  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_initializer= glorot_uniform(
  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(stage)+'_identity_1_b', ker
  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_c', kernel_initializer= gl
  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 "identity")
  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_initializer= glorot_uniform(
  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(stage)+'_identity_2_b', ker
  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_c', kernel_initializer= gl
  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 "identity")
  X = Add()([X,X_copy])
  X = Activation('relu')(X)

  return X
```

In [61]:
```
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(seed = 0))(X)
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)

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

# 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()
```

res_2_conv_b (Conv2D)	(None, 11, 11, 64)	36,928	activation_1[0][…
bn_2_conv_b (BatchNormalizatio…	(None, 11, 11, 64)	256	res_2_conv_b[0][…
activation_2 (Activation)	(None, 11, 11, 64)	0	bn_2_conv_b[0][0]
res_2_conv_copy (Conv2D)	(None, 23, 23, 256)	16,640	max_pooling2d[0]…
res_2_conv_c (Conv2D)	(None, 11, 11, 256)	16,640	activation_2[0][…
max_pooling2d_2 (MaxPooling2D)	(None, 11, 11, 256)	0	res_2_conv_copy[…
bn_2_conv_c (BatchNormalizatio…	(None, 11, 11, 256)	1,024	res_2_conv_c[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.
- What did you observe? Comment on your answer

The trainable parameter increases.

TASK #8: COMPILE AND TRAIN DEEP LEARNING MODEL

In [62]:
```
adam = tf.keras.optimizers.Adam(learning_rate = 0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)
model.compile(loss="mean_squared_error", optimizer = adam, metrics = ['accuracy'])
```

In [63]:
```
# save the best model with least validation loss
checkpointer = ModelCheckpoint(filepath = "weights.keras", verbose = 1, save_best_only = True)
```

In [64]:
```
history = model.fit(X_train, y_train, batch_size = 256, epochs= 100, validation_split = 0.05, callbacks=[checkpointe
```

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 15s 576ms/step - accuracy: 0.7896 - loss: 12.5996 - val_accuracy: 0.7493 - val_loss:
57.8743
Epoch 63/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 583ms/step - accuracy: 0.7989 - loss: 6.4842
Epoch 63: val_loss did not improve from 48.74915
26/26 ━━━━━━━━━━━━━━━━━━━━ 15s 591ms/step - accuracy: 0.7988 - loss: 6.4946 - val_accuracy: 0.7434 - val_loss: 5
9.7069
Epoch 64/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 571ms/step - accuracy: 0.7993 - loss: 7.2979
Epoch 64: val_loss did not improve from 48.74915
26/26 ━━━━━━━━━━━━━━━━━━━━ 15s 579ms/step - accuracy: 0.7992 - loss: 7.2736 - val_accuracy: 0.7347 - val_loss: 5
9.4505
Epoch 65/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 587ms/step - accuracy: 0.8092 - loss: 6.1864
Epoch 65: val_loss did not improve from 48.74915
26/26 ━━━━━━━━━━━━━━━━━━━━ 15s 594ms/step - accuracy: 0.8092 - loss: 6.1879 - val_accuracy: 0.7376 - val_loss: 6
4.1360
Epoch 66/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 573ms/step - accuracy: 0.8101 - loss: 7.1215
```

In [65]:
```
model_json = model.to_json()
with open('KeyPointDetector.json', 'w') as json_file:
    json_file.write(model_json)
```

MINI CHALLENGE:

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

In [66]:
```
history = model.fit(X_train, y_train, batch_size = 256, epochs= 50, validation_split = 0.1, callbacks=[checkpointer]
```

```
Epoch 1/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 560ms/step - accuracy: 0.7891 - loss: 11.4058
Epoch 1: val_loss improved from 45.09139 to 29.24899, saving model to weights.keras
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 580ms/step - accuracy: 0.7893 - loss: 11.3766 - val_accuracy: 0.7504 - val_loss:
29.2490
Epoch 2/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 562ms/step - accuracy: 0.7984 - loss: 8.0884
Epoch 2: val_loss improved from 29.24899 to 28.31678, saving model to weights.keras
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 584ms/step - accuracy: 0.7984 - loss: 8.0680 - val_accuracy: 0.8029 - val_loss: 2
8.3168
Epoch 3/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 573ms/step - accuracy: 0.8006 - loss: 6.1080
Epoch 3: val_loss did not improve from 28.31678
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 586ms/step - accuracy: 0.8005 - loss: 6.1030 - val_accuracy: 0.8058 - val_loss: 3
6.1507
Epoch 4/50
23/25 ━━━━━━━━━━━━━━━━━━━━ 1s 578ms/step - accuracy: 0.7969 - loss: 7.2255

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
```

TASK #9: ASSESS TRAINED MODEL PERFORMANCE

In [67]:
```
# instead of training from scratch, you can load traind model
with open('KeyPointDetector.json', 'r') as json_file:
    json_SavedModel = json_file.read()
model = tf.keras.models.model_from_json(json_SavedModel)
model.load_weights('weights.keras')
model.compile(loss='mean_squared_error', optimizer = adam, metrics = ['accuracy'])
```

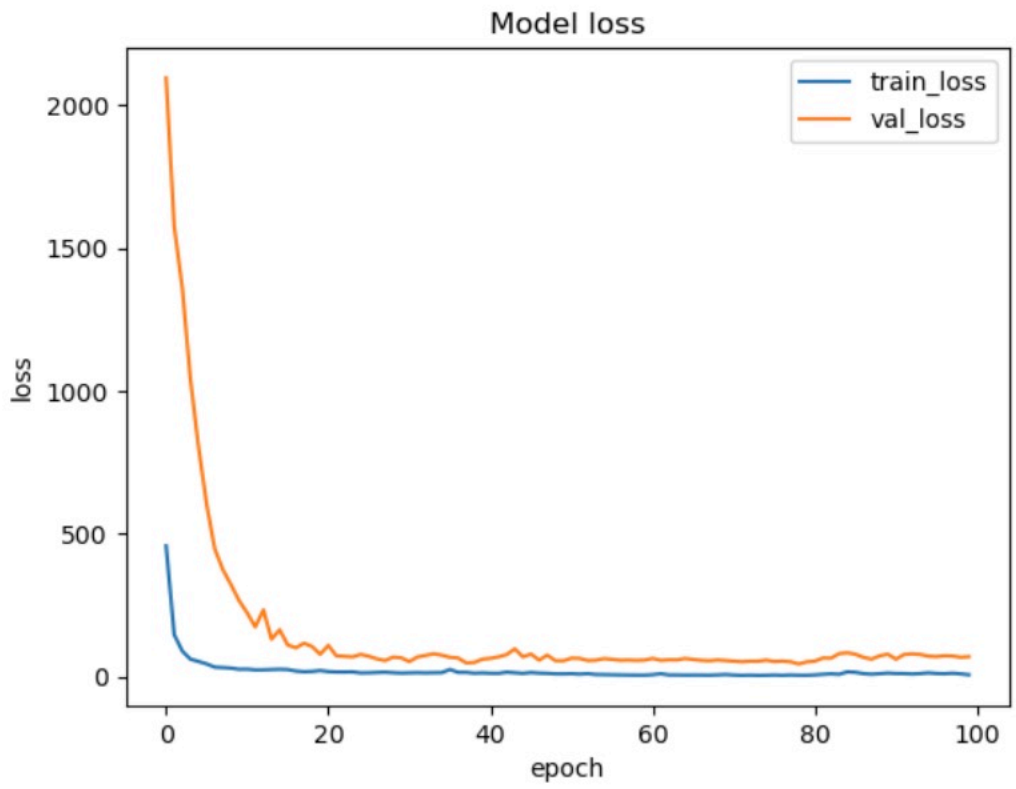
In [68]:
```
# Evaluate trained model

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

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.8078 - loss: 32.7351
Accuracy : 0.8037382960319519
```

In [69]:
```
# Make prediciton using the testing dataset
df_predict = model.predict(X_test)
```

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step
```

In [70]:
```
# 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 [71]:
```
# Make prediction using the testing dataset
df_predict = model.predict(X_test)
```

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step
```

In [72]:
```
# 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 : 5.8315857949018275
```

In [73]:
```
# Convert the predicted values into a dataframe

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

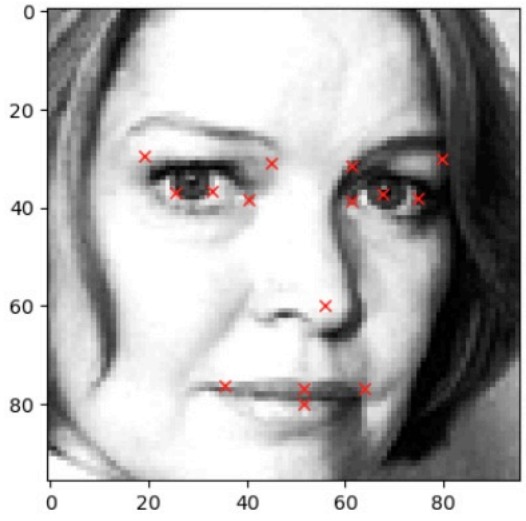
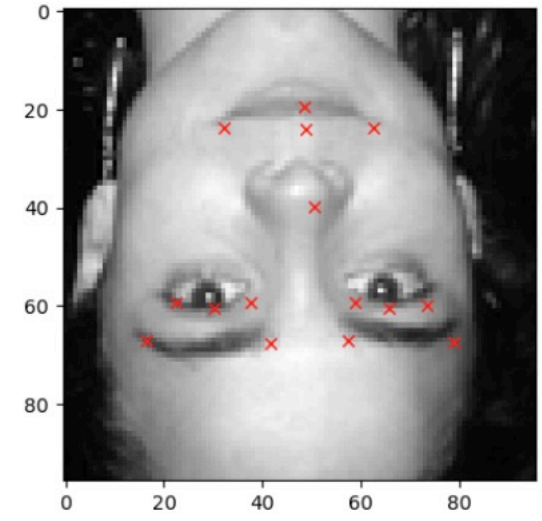
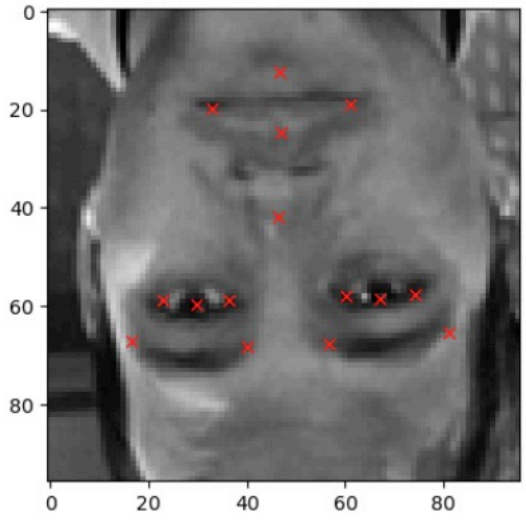
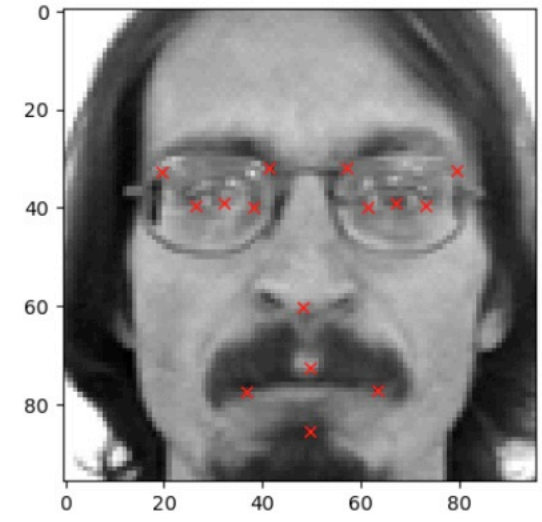
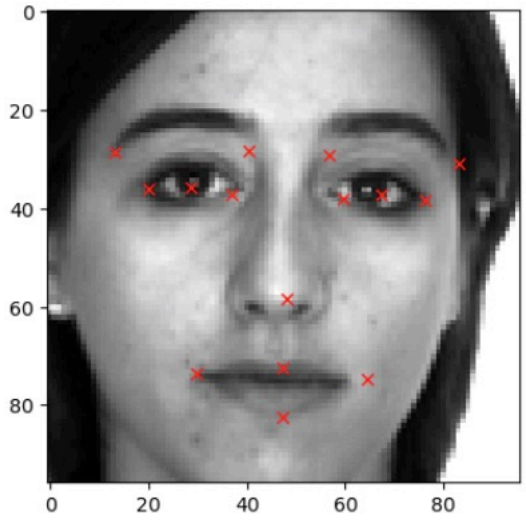
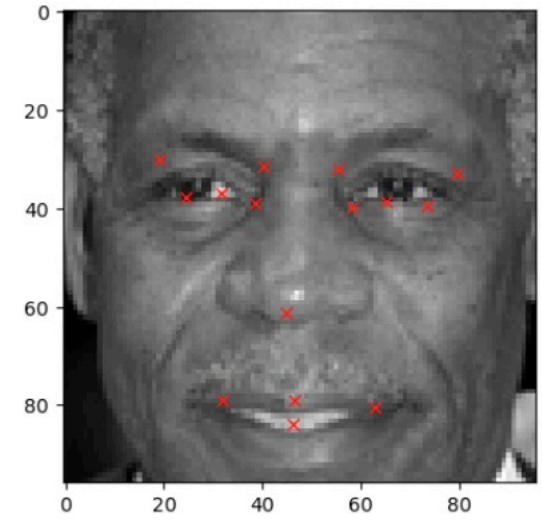
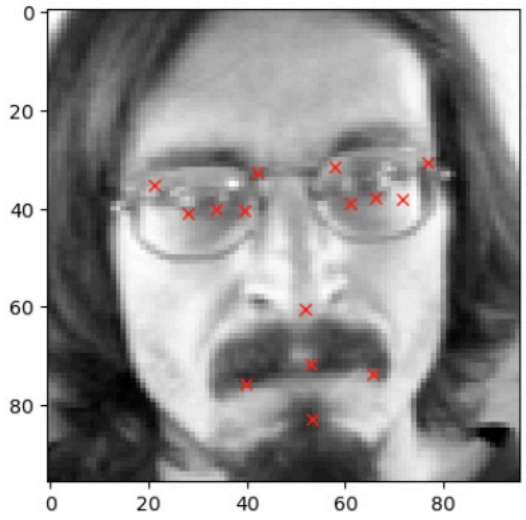
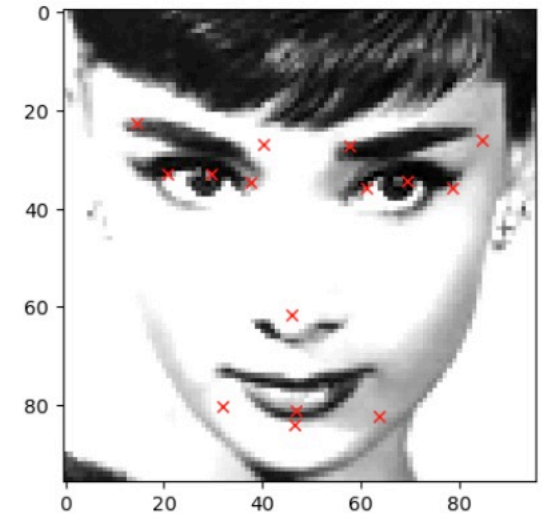
Out[73]:

	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_
0	67.695793	37.243301	32.960587	36.753716	61.361668	38.818531	74.654793	
1	65.892952	60.442749	30.153337	60.503197	58.858162	59.261158	73.394630	
2	67.105568	58.641563	29.774446	59.725758	60.285866	58.053925	74.090477	
3	67.105568	39.111187	32.343121	39.166466	61.333214	39.976742	73.330307	
4	67.336937	36.996059	28.630301	35.615520	59.583412	37.962231	76.146835	

5 rows × 30 columns

In [74]:
```
# 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')
```

```
/var/folders/83/bqlfyrvj1319dw3cqtd4cslc0000gn/T/ipykernel_2349/2024047750.py:10: FutureWarning: Series.__getitem__
treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consi
stent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  plt.plot(df_predict.loc[i][j-1], df_predict.loc[i][j], 'rx')
```

CONGRATULATIONS ON FINISHING THE PROJECT