

# **Gender Classification Using Convolutional Neural Networks and Decision Learning Tree**

**Artificial Intelligence**

**CS-E**

## **Group Members**

- 1. 17L-4010 (E)**
- 2. 17L-4154 (E)**
- 3. 17L-4238 (E)**
- 4. 17L-6326 (E)**

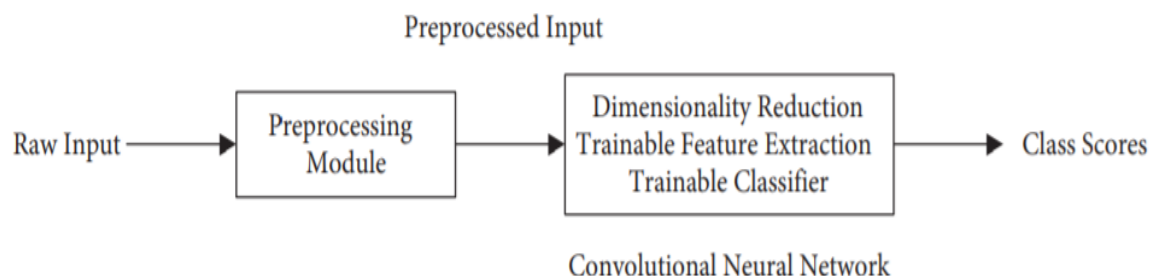
## Overview:

Face is one of the most imperative biometric attributes. Thus, face evaluation can help to gather great information such as gender. Therefore, a successful gender classification can help us boost performance in certain applications such as face recognition. To understand information regarding male/female characteristics is an interesting task with an additional benefit of contributing to the community of data science and machine learning.

For the purpose of this gender recognition, many deep neural networks can be used. Local deep neural networks have also been recently used. For our project, an approach using a convolutional neural network (CNN) is proposed for real-time gender classification based on facial images is used. In addition to the CNN classifier, we have also used the decision tree classifier for the gender classification and we then compare the efficiency of both the classifiers.

## CNN:

Convolutional neural network (or CNN) is a special type of multilayer neural network or deep learning architecture inspired by the visual system of living beings. CNN is very much suitable for different fields of computing vision.

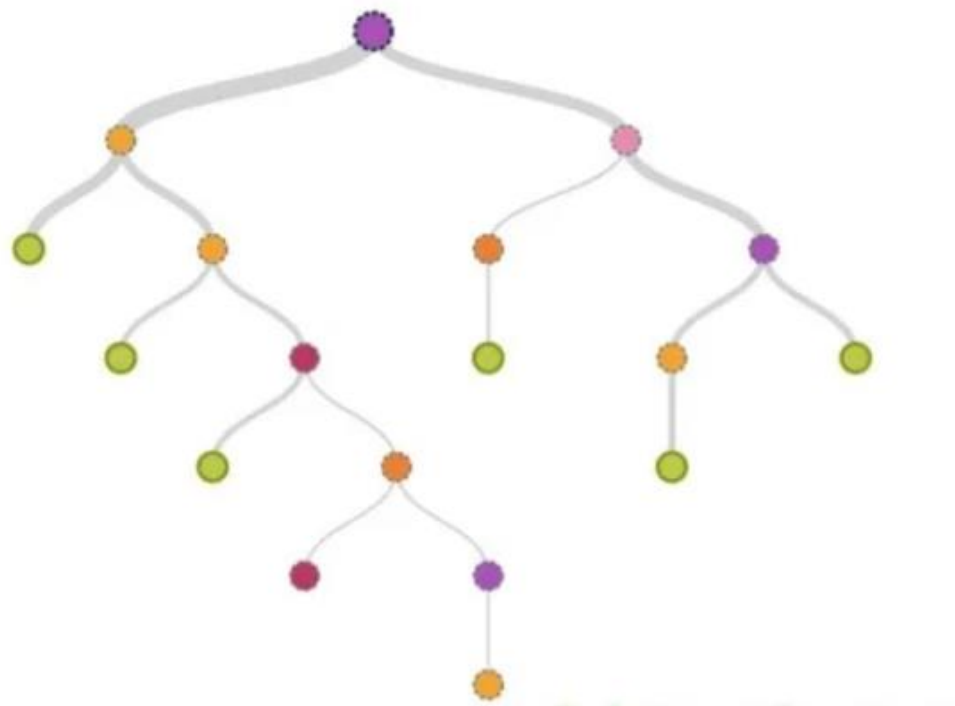


## Decision tree classifiers:

Decision tree classifiers give a meaningful classification model that is conceivably precise in various applications. The decision tree classifier creates the classification order model by building a choice tree. Every node in the tree determines a test on an attribute, each branch descends from that node relates to one of the possible value of the attributes. Cases in the training set are characterized by exploring them from the foundation of the tree down to a leaf, as indicated by the result of the tests along the way.

# Decision Tree Classifier

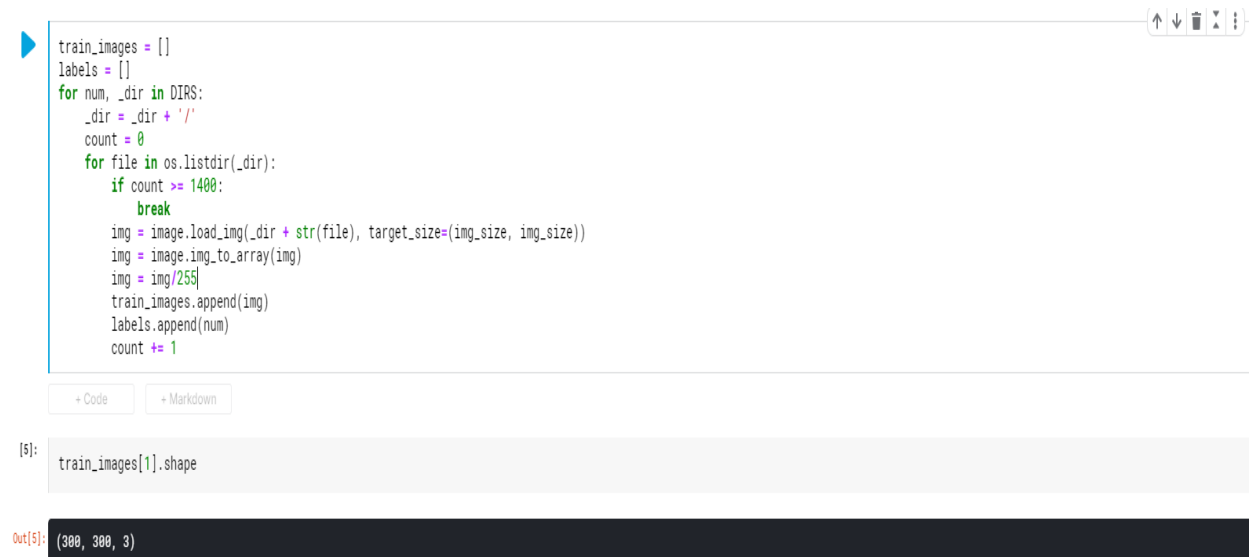
## In Python with Scikit-Learn



## Preprocessing:

The dataset which we used for both models consisted of a huge collection of images. The dataset is the “Gender Classification Dataset (Male Female Image Dataset)”. It contains 46,000 images( 23,000 for each class) for training the model and 11,000 5500 images of each class(men and women) for testing the model. The dataset is taken from the following:

[“Gender Classification Dataset”](#)



```
train_images = []
labels = []
for num, _dir in DIRS:
    _dir = _dir + '/'
    count = 0
    for file in os.listdir(_dir):
        if count >= 1400:
            break
        img = image.load_img(_dir + str(file), target_size=(img_size, img_size))
        img = image.img_to_array(img)
        img = img/255
        train_images.append(img)
        labels.append(num)
        count += 1
```

+ Code + Markdown

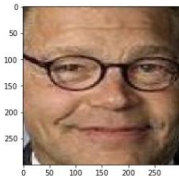
[5]: train\_images[1].shape

Out[5]: (300, 300, 3)

The images were trained to be of size 300x300. Examples are provided below.

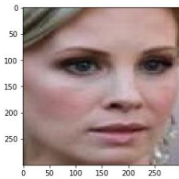
```
[6]: plt.imshow(train_images[4])
```

```
Out[6]: <matplotlib.image.AxesImage at 0x7f9e58dd62d8>
```



```
[7]: plt.imshow(train_images[1510])
```

```
Out[7]: <matplotlib.image.AxesImage at 0x7f9e58e45518>
```



In order to test and train our models, we decided to use 2800 images because our computing capacity wasn't capable enough to train 46000 images. In addition, there was the factor of time limitation as well. Training 2016 images on an epoch approximately took 15-20 minutes. Hence, we decided to choose data set of 2800 images.

2520 Images for Training Data

280 Images for Testing Data

Then we split the data.

Four variables used are

`X_train(2520), y_train_labels(2520,2), X_test(280), y_test(280,2)`

```
[8]: len(train_images)
```

```
Out[8]: 2800
```

```
[9]: X = np.array(train_images)
```

```
[10]: X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.1, random_state=101)
      print(X_train.shape)
      print(X_test.shape)
```

```
(2520, 300, 300, 3)
(280, 300, 300, 3)
```

```
[11]: len(X_train)
```

```
Out[11]: 2520
```

```
[12]: len(X_test)
```

```
Out[12]: 280
```

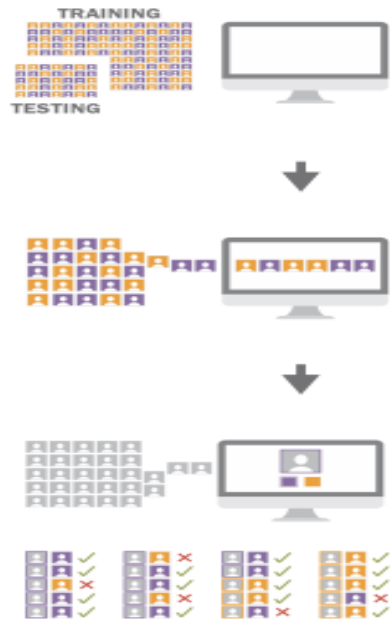
## Models Description:

The Simple Process we used was

- To select 2520 images and train our models on that data using fit command
- Once the models are trained, we then tested the (280) images using both CNN and Decision Tree Classifiers.
- Finally, we predicted the possible outcome of both models.

---

## Training a machine learning system to identify men and women in digital images



### Setting up

Researchers began with an untrained software system and a collection of images of faces that human coders labeled by gender. Some of the images were used to train the system, and some were set aside to later test the system and evaluate its ability to identify gender.

### Training

Researchers then showed the system the labeled training set of images, allowing it to develop its own rules for identifying images of men and women.

### Testing

Once the system was trained, researchers showed it the testing set of images, but with the human-coded labels hidden. The system then labeled each image as male or female. To evaluate the system's performance, researchers compared the system's decisions to those of the human coders.

"The Challenges of Using Machine Learning to Identify Gender in Images"

PEW RESEARCH CENTER

---

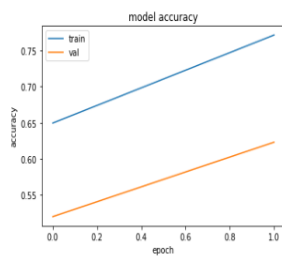
- CNN

```
=====
Total params: 6,954,498
Trainable params: 6,949,250
Non-trainable params: 5,248
=====
```

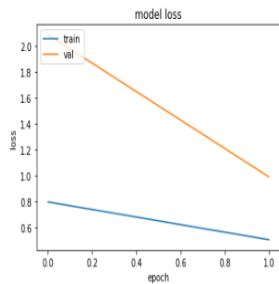
```
[18]: history = model.fit(X_train, y_train_labels, batch_size=16, epochs=2, validation_split=0.2)
```

```
Train on 2016 samples, validate on 504 samples
Epoch 1/2
2016/2016 [=====] - 722s 358ms/step - loss: 0.7987 - accuracy: 0.6496 - val_loss: 2.8887 - val_accuracy: 0.5198
Epoch 2/2
2016/2016 [=====] - 726s 360ms/step - loss: 0.5071 - accuracy: 0.7716 - val_loss: 0.9919 - val_accuracy: 0.6230
```

```
[19]: plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



```
[20]: plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



+ Code

+ Markdown

```
[21]: predictions = model.predict_classes(X_test)
print(predictions)
```

```
[0 0 1 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1
0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 0
0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0
0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0]
```



The graphs above show the accuracy of the training data as well as the testing data.

- Decision Tree Classifier

```
[34]: ##
      dtc_clf = dtc_clf.fit(X_test, y_test)
      dtc_prediction = dtc_clf.predict(X_test)
      print (dtc_prediction)
```

```
[0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 1 0 1 1 0 1 1 1 1 1 0 0 1 1 0
 1 1 0 1 0 1 1 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0 1 1 1 0 0 0 0 1 0 1 1 0 1
 0 1 0 0 0 1 0 0 0 0 0 1 1 0 1 0 1 0 1 0 0 1 1 1 1 1 0 1 0 1 0 1 1 1 1 0 0
 0 1 1 1 1 0 0 0 1 0 0 0 1 0 1 0 1 1 1 0 1 0 0 1 0 1 1 1 0 0 0 1 1 0 0
 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 0 0 1 0 0 1 0
 0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 0 1 0 0 0 0 1 1 0 1 1 0 0 0 0 0
 1 0 0 0 0 1 1 0 1 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 0 0 0 0 1 1 1 0 1 0 1 1 1
 1 1 0 0 1 0 1 1 1 1 0 1 1 0 1 0 0 1 0 1 1]
```

A comparison is drawn between the performance of both models using three main concepts and tools which are as follows.

- Confusion Matrix: In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).
- Classification Report
- Set of Random Images from Dataset

- CNN

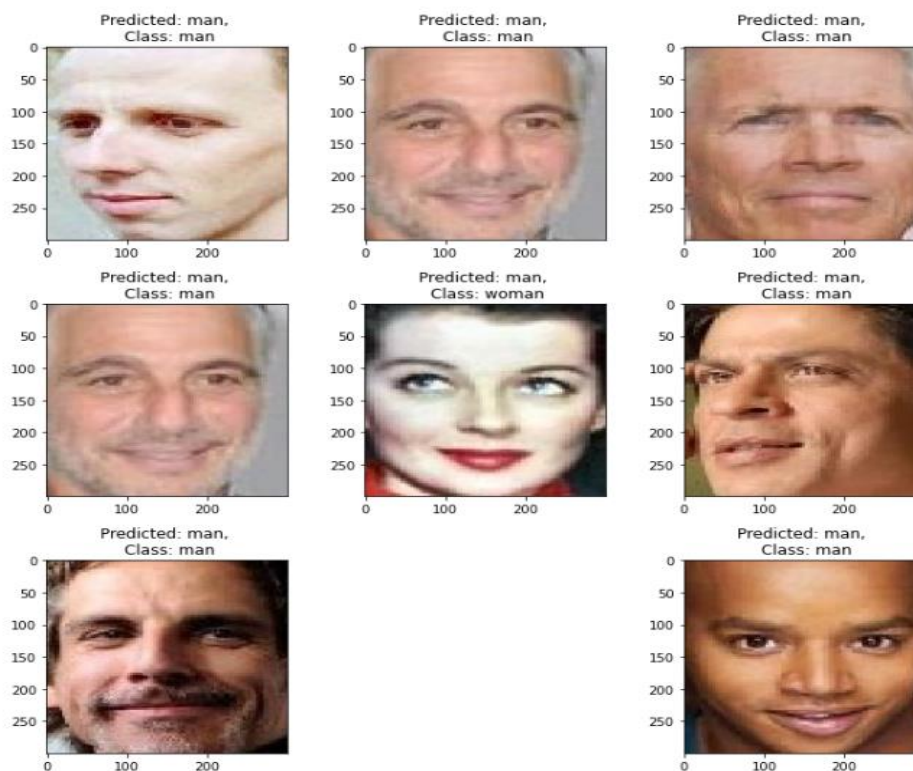
Confusion Matrix shows that out of 280 images, 148 were male images and 132 were female images. In the case of CNN, 134 male images were labeled correctly while the model wrongly predicted 14 as female images, and out of 132 female images, 51 were predicted correctly while the model wrongly predicted 81 as male images. Furthermore, Classification report shows that the CNN model was able to produce an accuracy of 0.66 equivalent to 66%.

```
[22]: print(confusion_matrix(predictions, y_test))
```

```
[[134  81]
 [ 14  51]]
```

```
[23]: print(classification_report(predictions, y_test))
```

	precision	recall	f1-score	support
0	0.91	0.62	0.74	215
1	0.39	0.78	0.52	65
accuracy			0.66	280
macro avg	0.65	0.70	0.63	280
weighted avg	0.78	0.66	0.69	280



In the above diagram, we can conclude that the CNN model was able to predict 7 out of 8 images correctly. All these images were randomly selected from the set of test data. Therefore, In some other cases, It is possible that the model can predict more inaccurately.

- **Decision Tree Classifier**

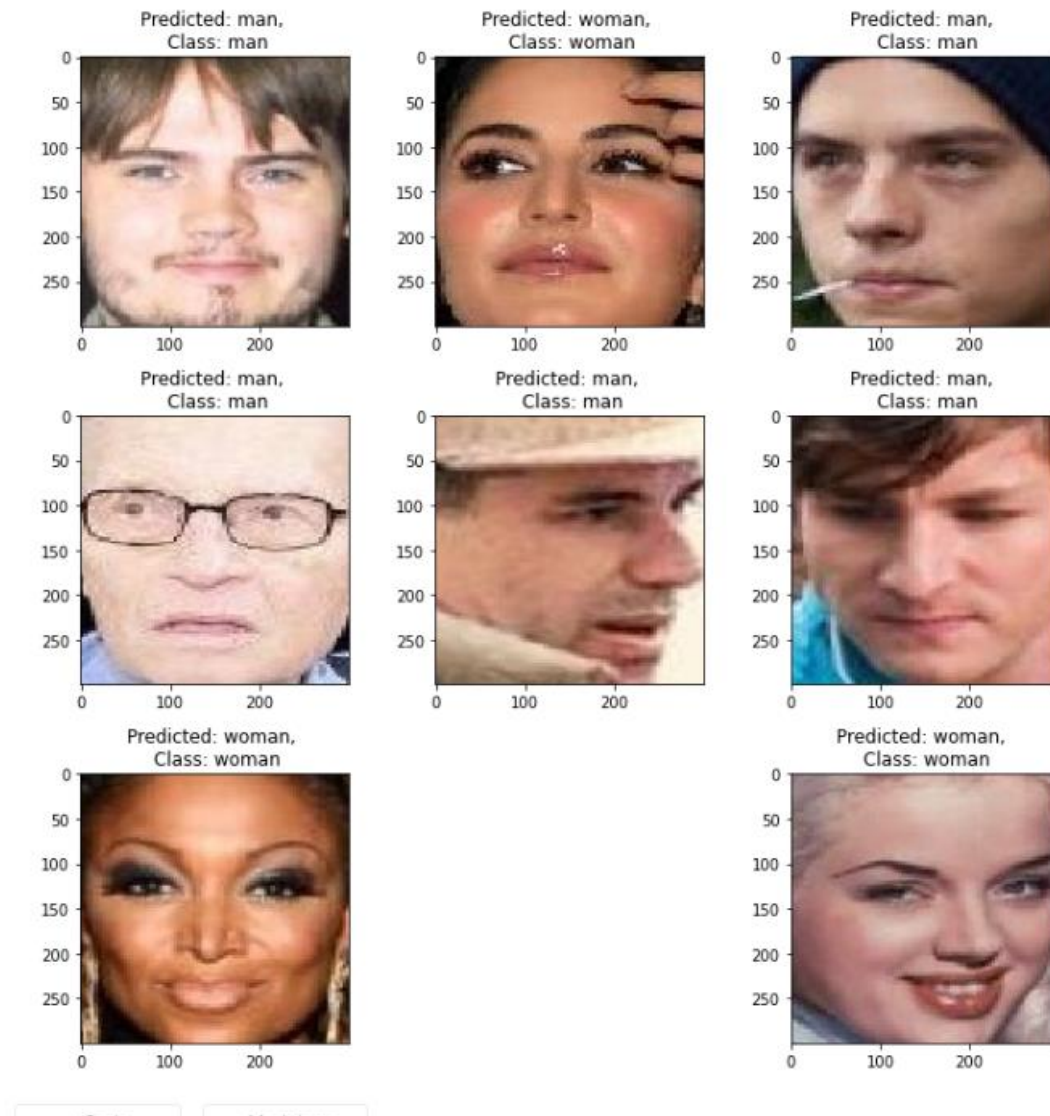
Confusion Matrix shows that out of 280 images, 148 were male images and 132 were female images. In the case of Decision Tree Classifier, 148 male images were labeled correctly while the model wrongly predicted 0 as female images and out of 132 female images, 132 were predicted correctly while the model wrongly predicted 0 as male Images. Furthermore, Classification report shows that the CNN model was able to produce an accuracy of 1.0 equivalent to 100%.

```
[35]: #AH
      print(confusion_matrix(dtc_prediction, y_test))
```

```
[[148  0]
 [  0 132]]
```

```
[36]: print(classification_report(dtc_prediction, y_test))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	148
1	1.00	1.00	1.00	132
accuracy			1.00	280
macro avg	1.00	1.00	1.00	280
weighted avg	1.00	1.00	1.00	280



In the above diagram, we can conclude that the Decision Tree Classifier model was able to predict 8 out of 8 images correctly. All these images were randomly selected from the set of test data. Therefore, in some other case, It model will also produce accurate classification.

### Final Remarks:

In terms of performance, Decision Tree was able to produce much better classification as compared to CNN on the dataset that we have used. However, the efficiency of the CNN could be improved by increasing the number of epochs or the number of training images.