Gender Classification based on eye

There has been several studies based on this project!

- One study is based on Binary Statistical Features (BSIF) algorithm for classifying gender from iris texture images captured with NIR sensors. It uses the same pipeline for iris recognition systems consisting of iris segmentation, normalisation and then classification. Experiments show that applying BSIF is not straightforward since it can create artificial textures causing misclassification. In order to overcome this limitation, a new set of filters was trained from eye images and different sized filters with padding bands were tested on a subject-disjoint database. A Modified-BSIF (MBSIF) method was implemented. The latter achieved better gender classification results (94.6% and 91.33% for the left and right eye respectively). These results are competitive with the state of the art in gender classification. In an additional contribution, a novel gender labelled database was created and it will be available upon request.
- Another study is based on superior face recognition skills in females, partially due to their different eye movement strategies when encoding faces. In the current study, we utilized these slight but important differences and proposed a model that estimates the gender of the viewers and classifies them into two subgroups, males and females. An eye tracker recorded participant's eye movements while they viewed images of faces. Regions of interest (ROIs) were defined for each face. Results showed that the gender dissimilarity in eye movements was not due to differences in frequency of fixations in the ROI s per se. Instead, it was caused by dissimilarity in saccade paths between the ROIs. The difference enhanced when saccades were towards the eyes. Females showed significant increase in transitions from other ROI s to the eyes. Consequently, the extraction of temporal transient information of saccade paths through a transition probability matrix, similar to a first order Markov chain model, significantly improved the accuracy of the gender classification results.
- Another study is focused primarily on the use of texture features and not much research has been done on applying Convolutional Neural Networks (CNN) to this task. In this work we trained a small convolutional neural network for the left and right eyes, and more importantly, studied the effect of merging those models and compare it against the model obtained by training a CNN over the fused left-right eye images. We show that the network benefits from this model merging approach, and becomes more robust towards occlusion and low resolution degradation, outperforming the results of using a single CNN model for the left and right set of images. Experiments done over a database of near-infrared periocular images show that our CNN model exhibits competitive performance compared to other state-of-the-art methods.

For the same purpose, we have also used Convolutional Neural Network for our task.

Framework and Dataset

Framework: Keras has been used in this project, Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

Dataset: The dataset used in this project has been taken from an opensource sight. this dataset includes:

Number of images: 11525

Number of images with malee eyes: 6320

Number of images with femalee eyes: 5202

Importing Required Libraries

```
import pandas as pd
import numpy as np
import datetime as dt
import os
import os.path
from pathlib import Path
import glob
import cv2
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, BatchNorma
from tensorflow.keras import layers
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.xception import Xception
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.mobilenet import MobileNet
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
from sklearn.metrics import confusion_matrix, classification_report, recall_score, precision_
from tensorflow.keras.preprocessing import image
from PIL import Image
```

```
from google.colab import drive
drive.mount('/drive')
```

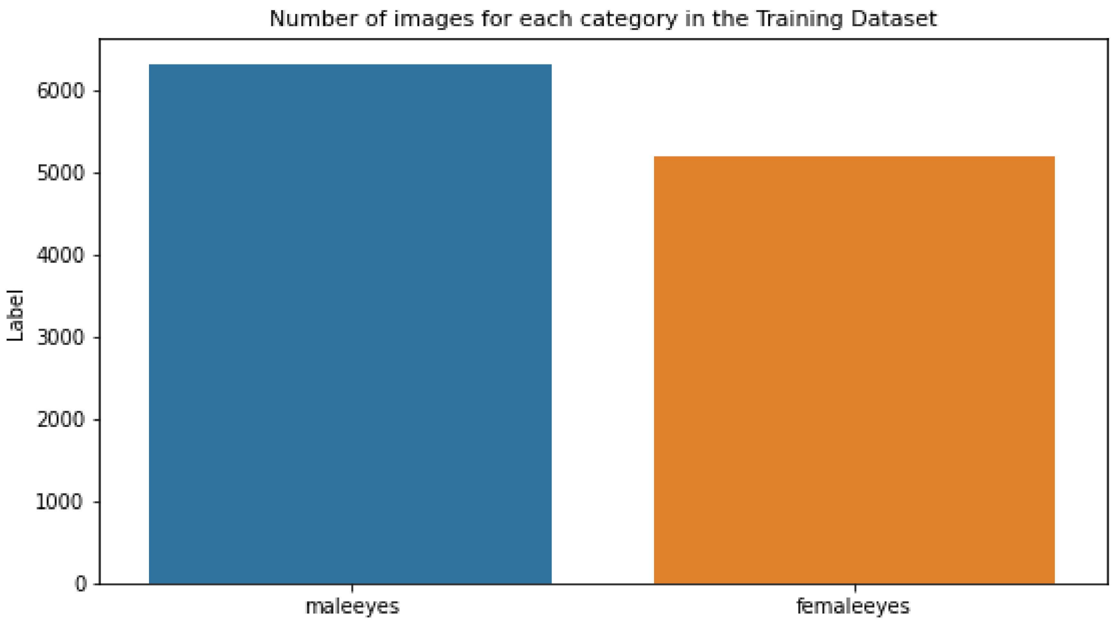
```
Drive already mounted at /drive; to attempt to forcibly remount, call drive.mount("/driv
```

ARRANGING MALE AND FEMALE DATASET

```
# Selecting Dataset Folder Paths
f_dir_ = Path('/drive/MyDrive/input/femaleeyes')
m_dir_ = Path('/drive/MyDrive/input/maleeyes')

femaleeyes_filepaths = list(f_dir_.glob(r'**/*.jpg'))
maleeyes_filepaths = list(m_dir_.glob(r'**/*.jpg'))
print(femaleeyes_filepaths)
# Mapping the labels
fm_labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1], femaleeyes_filepaths))
ml_labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1], maleeyes_filepaths))

# Paths & labels femalee eyes
fm_filepaths = pd.Series(femaleeyes_filepaths, name = 'File').astype(str)
fm_labels = pd.Series(fm_labels, name='Label')

# Paths & labels malee eyes
ml_filepaths = pd.Series(maleeyes_filepaths, name = 'File').astype(str)
ml_labels = pd.Series(ml_labels, name='Label')

# Concatenating...
femaleeyes_df = pd.concat([fm_filepaths, fm_labels], axis=1)
maleeyes_df = pd.concat([ml_filepaths, ml_labels], axis=1)

df = pd.concat([femaleeyes_df, maleeyes_df])

df = df.sample(frac = 1, random_state = 56).reset_index(drop = True)
vc = df['Label'].value_counts()

plt.figure(figsize = (9, 5))
sns.barplot(x = vc.index, y = vc)
```

```
    sns.barplot(x = vc.index, y = vc)
    plt.title("Number of images for each category in the Training Dataset", fontsize = 11)
    plt.show()
```

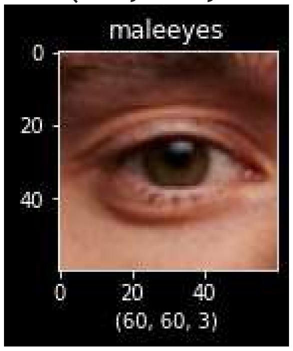
```
[PosixPath('/drive/MyDrive/input/femaleeyes/955.jpg'), PosixPath('/drive/MyDrive/input/f
```



Observing the MALE AND FEMALE images

```
plt.style.use("dark_background")

figure = plt.figure(figsize=(2,2))
x = plt.imread(df["File"][34])
plt.imshow(x)
plt.xlabel(x.shape)
plt.title(df["Label"][34])
```

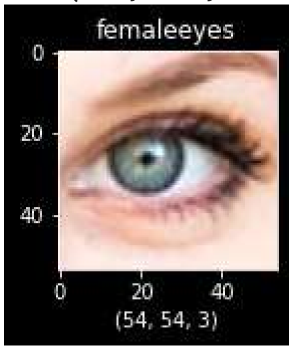
```
Text(0.5, 1.0, 'maleeyes')
```



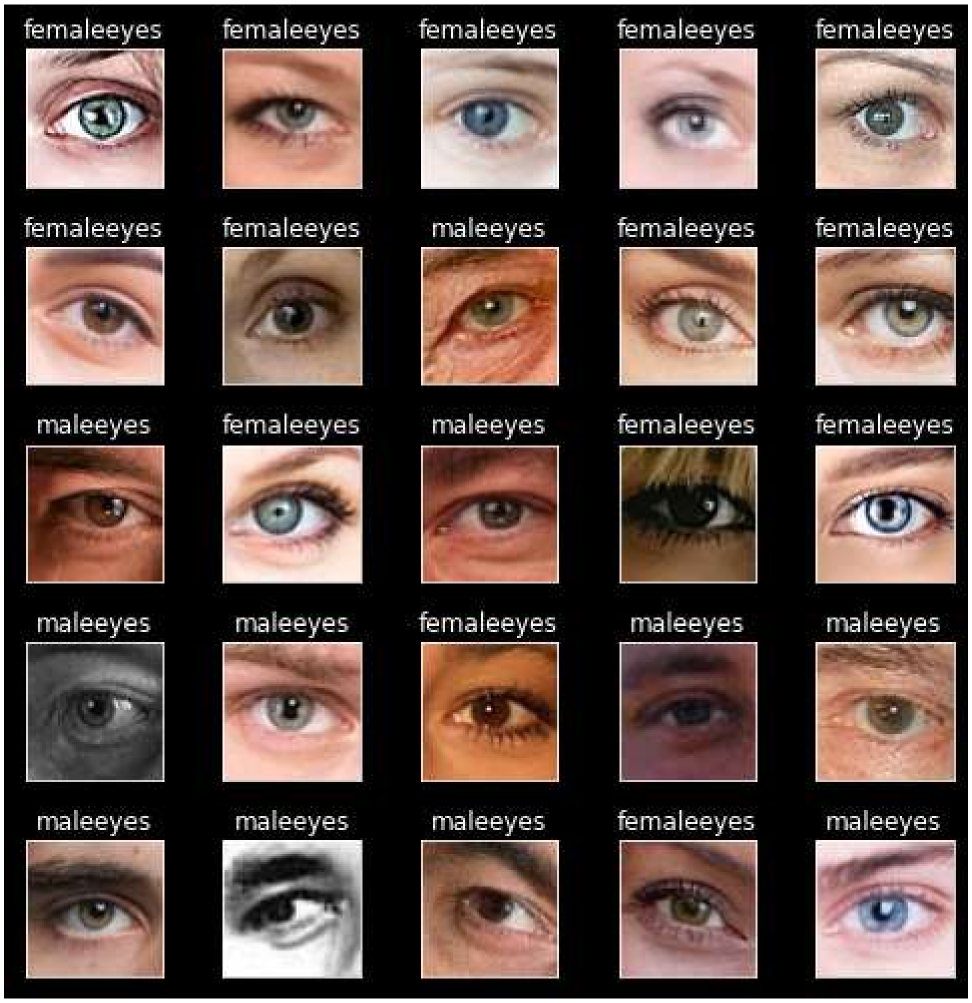
```
figure = plt.figure(figsize=(2, 2))
x = plt.imread(df["File"][11])
plt.imshow(x)
plt.xlabel(x.shape)
plt.title(df["Label"][11])
```

```
    plt.title(df["Label"][ii])
```

```
        Text(0.5, 1.0, 'femaleeyes')
```



```
    fig, axes = plt.subplots(nrows = 5,
                             ncols = 5,
                             figsize = (7, 7),
                             subplot_kw = {"xticks":[],"yticks":[]})

    for i,ax in enumerate(axes.flat):
        ax.imshow(plt.imread(df["File"][i]))
        ax.set_title(df["Label"][i])
    plt.tight_layout()
    plt.show()
```



SPLITTING TRAINSET AND TESTSET WITH 75% AND 25% RESPECTIVELY

```
trainset_df, testset_df = train_test_split(df, train_size = 0.75, random_state = 4)

display(trainset_df.head())

testset_df.head()
```

	File	Label
5614	/drive/MyDrive/input/maleeyes/5269.jpg	maleeyes
862	/drive/MyDrive/input/femaleeyes/3246.jpg	femaleeyes
9030	/drive/MyDrive/input/femaleeyes/1592.jpg	femaleeyes
7482	/drive/MyDrive/input/maleeyes/3269.jpg	maleeyes
5000	/drive/MyDrive/input/maleeyes/5232.jpg	maleeyes

	File	Label
3814	/drive/MyDrive/input/femaleeyes/2505.jpg	femaleeyes
5238	/drive/MyDrive/input/femaleeyes/5782.jpg	femaleeyes
2768	/drive/MyDrive/input/femaleeyes/4819.jpg	femaleeyes
2398	/drive/MyDrive/input/maleeyes/3061.jpg	maleeyes
9093	/drive/MyDrive/input/maleeyes/1333.jpg	maleeyes

Converting the Label to a numeric format for testing later...

```
LE = LabelEncoder()

y_test = LE.fit_transform(testset_df["Label"])
# Viewing data in training dataset
print('Training Dataset:')

print(f'Number of images: {trainset_df.shape[0]}')

print(f'Number of images with malee eyes: {trainset_df["Label"].value_counts()[0]}')
print(f'Number of images with femalee eyes: {trainset_df["Label"].value_counts()[1]}\n')

# Viewing data in test dataset
print('Test Dataset:')

print(f'Number of images: {testset_df.shape[0]}')
print(f'Number of images: {testset_df.shape[0]}')

print(f'Number of images with malee eyes: {testset_df["Label"].value_counts()[0]}')
print(f'Number of images with femalee eyes: {testset_df["Label"].value_counts()[1]}\n')
```

```
     Training Dataset:
     Number of images: 8643
     Number of images with malee eyes: 4729
     Number of images with femalee eyes: 3914

     Test Dataset:
     Number of images: 2882
     Number of images with malee eyes: 1594
     Number of images with femalee eyes: 1288
```

Generating batches of images

Let's generate batches of images increasing the training data, for the test database we will just normalize the data using ImageDataGenerator

Parameters of ImageDataGenerator:

```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.1,
                                   rotation_range = 20,
                                   width_shift_range = 0.1,
                                   height_shift_range = 0.1,
                                   horizontal_flip = True,
                                   vertical_flip = True,
                                   validation_split = 0.1)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

Directory of training, validation and test images

Divide the image bases for training, validation and testing of the model, for that we use the flow_from_dataframe

Parameters of flow_from_directory:

```
'''dataframe - Dataframe containing the images directory
x_col - Column name containing the images directory
y_col - Name of the column containing what we want to predict
target_size - size of the images (remembering that it must be the same size as the input laye
color_mode - RGB color standard
class_mode - binary class mode (cat/dog)
class_mode - binary class mode (cat/dog)
batch_size - batch size (32)
shuffle - Shuffle the data
seed - optional random seed for the shuffle
subset - Subset of data being training and validation (only used if using validation_split in
'''
```

```
     'dataframe - Dataframe containing the images directory\nx_col - Column name containing
     the images directory\ny_col - Name of the column containing what we want to predict\nta
     rget_size - size of the images (remembering that it must be the same size as the input
     layer)\ncolor_mode - RGB color standard\nclass_mode - binary class mode (cat/dog)\nbatc
```

```
print("Preparing the training dataset ...")
training_set = train_datagen.flow_from_dataframe(
    dataframe = trainset_df,
    x_col = "File",
    y_col = "Label",
    target_size = (75, 75),
    color_mode = "rgb",
    class_mode = "binary",
    batch_size = 32,
    shuffle = True,
    seed = 2,
    subset = "training")

print("Preparing the validation dataset ...")
validation_set = train_datagen.flow_from_dataframe(
    dataframe = trainset_df,
    x_col = "File",
    y_col = "Label",
    target_size = (75, 75),
    color_mode ="rgb",
    class_mode = "binary",
    batch_size = 32,
    shuffle = True,
    seed = 2,
    subset = "validation")

print("Preparing the test dataset ...")
test_set = test_datagen.flow_from_dataframe(
    dataframe = testset_df,
    x_col = "File",
    y_col = "Label",
    target_size = (75, 75),
    color_mode ="rgb",
    class_mode = "binary",
    shuffle = False,
    batch_size = 32)

print('Data generators are ready!')
```

```
     Preparing the training dataset ...
     Found 7779 validated image filenames belonging to 2 classes.
```

```
Preparing the validation dataset ...
Found 864 validated image filenames belonging to 2 classes.
Preparing the test dataset ...
Found 2882 validated image filenames belonging to 2 classes.
Data generators are ready!
```

```
print("Training: ")
print(training_set.class_indices)
print(training_set.image_shape)
print("---" * 8)
print("Validation: ")
print(validation_set.class_indices)
print(validation_set.image_shape)
print("---" * 8)
print("Test: ")
print(test_set.class_indices)
print(test_set.image_shape)
# Callbacks
cb = [EarlyStopping(monitor = 'loss', mode = 'min', patience = 15, restore_best_weights = Tru
```

```
Training:
{'femaleeyes': 0, 'maleeyes': 1}
(75, 75, 3)
------------------------
Validation:
{'femaleeyes': 0, 'maleeyes': 1}
(75, 75, 3)
------------------------
Test:
{'femaleeyes': 0, 'maleeyes': 1}
(75, 75, 3)
```

```
'''monitor - Metrics that will be monitored
patience - Number of times without improvement in the model, after these times the training i
restore_best_weights - Restores best weights if training is interrupted
# Callbacks
cb = [EarlyStopping(monitor = 'loss', mode = 'min', patience = 15, restore_best_weights = Tru
```

```
'monitor - Metrics that will be monitored\npatience - Number of times without improveme
nt in the model, after these times the training is stopped\nrestore_best_weights - Rest
ores best weights if training is interrupted\n# Callbacks\ncb = [EarlyStopping(monitor
```

Construction of the first model (ConvNet)

CNNs are a specific type of artificial neural network that is very effective for image classification because they are able to take into account the spatial coherence of the image, that is, that pixels close to each other are often related.

The construction of a CNN begins with specifying the model type. In our case, we will use a Sequential model.

What is Neural Network A neural network is simply a group of connected neurons, there are some input neurons, some output neurons and a group of what we call hidden neurons in between. When we feed information to the input neurons we get some information from the output neurons. Information starts at the input neurons and travels to the next layers of neurons having whats called a weight and a bias applied to it. These weight and biases start out randomly determined and are tweaked as the network learns and sees more data. After reaching a new layer there is a function applied to each neurons value that is called an activation function.

How does Neural Network work?

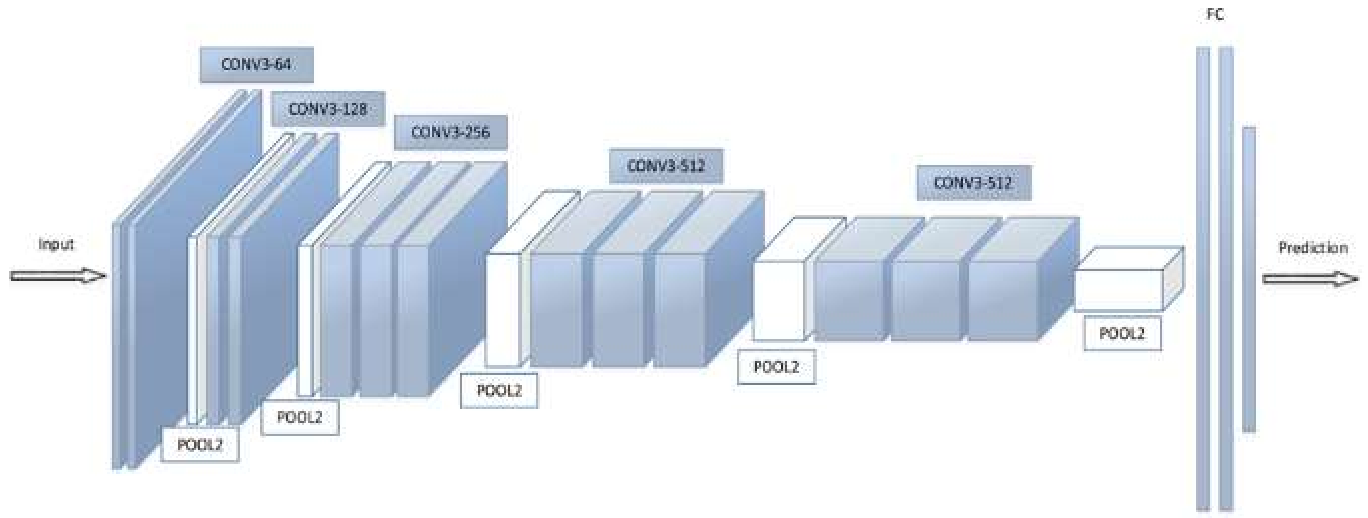
y=w1*x1+w2*x2+w3*x3+w4(bias) z=act(y)

First weights will be passed to hidden neurons, and two steps would happen.

1. The summation of weights and features would happen and then bias is added
2. Activation function is applied to the summation of weights

How does activation function work? Lets suppose, if we keep our one hand on a hot object, then the neurons of that hand would get activated, but not of the other hands

1. Sigmoid Activation Function: used in Logistic Regression, and Activation function is 1/(1+e^-y). Any value of y given to sigmoid would result either in 0 or in 1. There is not in between value.
2. ReLu Activation Function: the product and summation of weights, features, and bias is sent to Relu, and max(y, 0) is found in Relu.



Step 1 - Convolution

Feature Detector and Feature Map

Number of filters (32)

Dimensions of the feature detector (3, 3)

Definition of height / width and RGB channels (128, 128, 3)

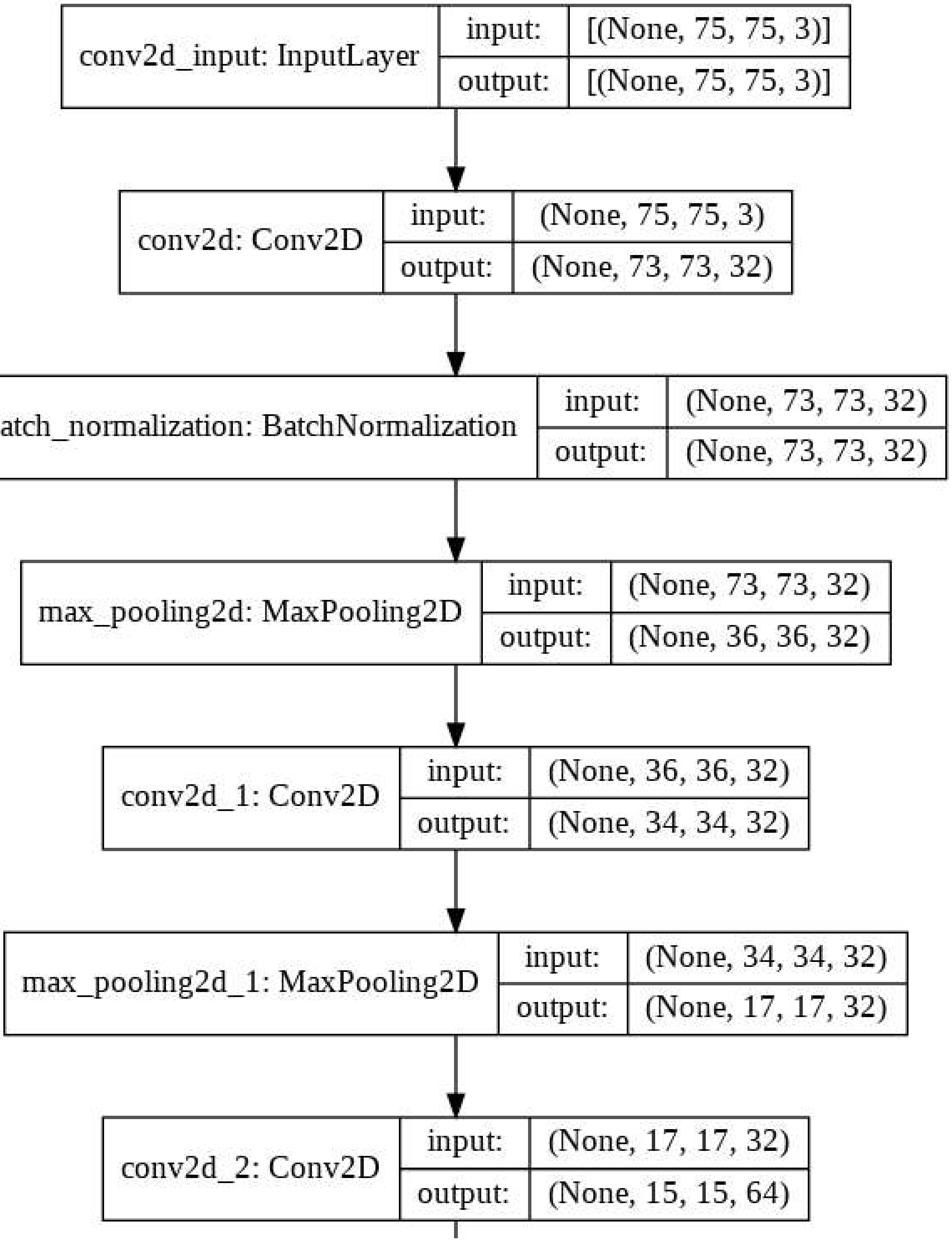
Activation function to remove negative values from the image - 'relu'

Processing acceleration - BatchNormalization

```
CNN = Sequential()

CNN.add(Conv2D(32, (3, 3), input_shape = (75, 75, 3), activation = 'relu'))
CNN.add(BatchNormalization())



CNN.add(MaxPooling2D(pool_size = (2, 2)))



CNN.add(Conv2D(32, (3, 3), activation = 'relu'))
CNN.add(MaxPooling2D(pool_size = (2, 2)))


CNN.add(Conv2D(64, (3, 3), activation = 'relu'))
CNN.add(SpatialDropout2D(0.2))
CNN.add(MaxPooling2D(pool_size = (2, 2)))


CNN.add(Flatten())
CNN.add(Dense(units = 128, activation = 'relu'))
CNN.add(Dropout(0.2))
# Output layer (binary classification)
CNN.add(Dense(units = 1, activation = 'sigmoid'))

print(CNN.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 73, 73, 32)        896       
_________________________________________________________________
batch_normalization (BatchNo (None, 73, 73, 32)        128       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 36, 36, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 34, 34, 32)        9248      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 17, 17, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 15, 15, 64)        18496     
_________________________________________________________________
spatial_dropout2d (SpatialDr (None, 15, 15, 64)        0         
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0         
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense (Dense)                (None, 128)               401536
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 430,433
Trainable params: 430,369
Non-trainable params: 64
_________________________________________________________________
None
```

```
plot_model(CNN, to_file='CNN_model.png', show_layer_names = True , show_shapes = True)
```


conv2d_input: InputLayer
input: [(None, 75, 75, 3)]
output: [(None, 75, 75, 3)]
conv2d: Conv2D
input: (None, 75, 75, 3)
output: (None, 73, 73, 32)
batch_normalization: BatchNormalization
input: (None, 73, 73, 32)
output: (None, 73, 73, 32)
max_pooling2d: MaxPooling2D
input: (None, 73, 73, 32)
output: (None, 36, 36, 32)
conv2d_1: Conv2D
input: (None, 36, 36, 32)
output: (None, 34, 34, 32)
max_pooling2d_1: MaxPooling2D
input: (None, 34, 34, 32)
output: (None, 17, 17, 32)
conv2d_2: Conv2D
input: (None, 17, 17, 32)
output: (None, 15, 15, 64)

Step 6 - Model compilation and training

Now that we have specified the model architecture, we will compile the model for training. For this, we need to specify the loss function (what we are trying to minimize), the optimizer (how we want to do to minimize the loss) and the metric (how we will judge the model's performance). Next, we will call .fit to start training the process

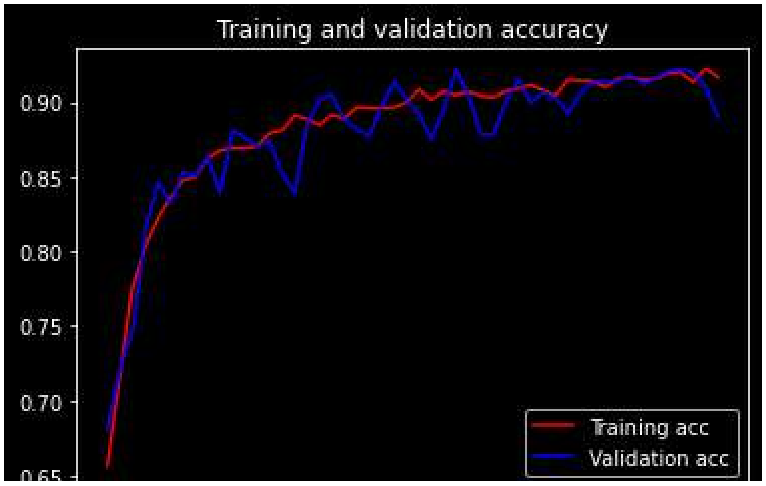
```
# Compile
CNN.compile(optimizer='adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
# Start of counting time...
start = dt.datetime.now()

# Train
CNN_model = CNN.fit(training_set, epochs = 50, validation_data = validation_set, callbacks =

# End of counting time...
end = dt.datetime.now()
time_CNN = end - start
print ('\nTraining and validation time is: ', time_CNN)
```

```
     Epoch 16/50
     244/244 [==============================] - 87s 355ms/step - loss: 0.2641 - accuracy:
     Epoch 17/50
     244/244 [==============================] - 87s 355ms/step - loss: 0.2718 - accuracy:
     Epoch 18/50
     244/244 [==============================] - 86s 351ms/step - loss: 0.2746 - accuracy:
     Epoch 19/50
     244/244 [==============================] - 85s 350ms/step - loss: 0.2620 - accuracy:
     Epoch 20/50
     244/244 [==============================] - 85s 348ms/step - loss: 0.2651 - accuracy:
     Epoch 21/50
     244/244 [==============================] - 85s 349ms/step - loss: 0.2511 - accuracy:
     Epoch 22/50
     244/244 [==============================] - 85s 348ms/step - loss: 0.2551 - accuracy:

     Epoch 23/50
     244/244 [==============================] - 85s 347ms/step - loss: 0.2542 - accuracy:
     Epoch 24/50
     244/244 [==============================] - 85s 350ms/step - loss: 0.2473 - accuracy:
     Epoch 25/50
     244/244 [==============================] - 86s 353ms/step - loss: 0.2440 - accuracy:
     Epoch 26/50
     244/244 [==============================] - 86s 353ms/step - loss: 0.2333 - accuracy:
     Epoch 27/50
     244/244 [==============================] - 86s 353ms/step - loss: 0.2381 - accuracy:
     Epoch 28/50
     244/244 [==============================] - 86s 352ms/step - loss: 0.2290 - accuracy:
     Epoch 29/50
     244/244 [==============================] - 86s 352ms/step - loss: 0.2414 - accuracy:
     Epoch 30/50
     244/244 [==============================] - 86s 353ms/step - loss: 0.2337 - accuracy:
     Epoch 31/50
     244/244 [==============================] - 86s 354ms/step - loss: 0.2354 - accuracy:
     Epoch 32/50
     244/244 [==============================] - 86s 354ms/step - loss: 0.2354 - accuracy:
     Epoch 33/50
     244/244 [==============================] - 86s 351ms/step - loss: 0.2259 - accuracy:
     Epoch 34/50
     244/244 [==============================] - 87s 354ms/step - loss: 0.2228 - accuracy:
     Epoch 35/50
     244/244 [==============================] - 87s 354ms/step - loss: 0.2145 - accuracy:
     Epoch 36/50
     244/244 [==============================] - 85s 348ms/step - loss: 0.2277 - accuracy:
     Epoch 37/50
     244/244 [==============================] - 84s 344ms/step - loss: 0.2351 - accuracy:
     Epoch 38/50
```

```
244/244 [==============================] - 84s 345ms/step - loss: 0.2181 - accuracy: (
Epoch 39/50
244/244 [==============================] - 85s 348ms/step - loss: 0.2159 - accuracy: (
Epoch 40/50
244/244 [==============================] - 85s 350ms/step - loss: 0.2148 - accuracy: (
Epoch 41/50
244/244 [==============================] - 85s 346ms/step - loss: 0.2282 - accuracy: (
Epoch 42/50
244/244 [==============================] - 85s 349ms/step - loss: 0.2147 - accuracy: (
Epoch 43/50
244/244 [==============================] - 86s 354ms/step - loss: 0.2133 - accuracy: (
Epoch 44/50
244/244 [==============================] - 86s 353ms/step - loss: 0.2165 - accuracy: (
Epoch 45/50
```

```
acc = CNN_model.history['accuracy']
val_acc = CNN_model.history['val_accuracy']
loss = CNN_model.history['loss']
val_loss = CNN_model.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.title('Training and validation accuracy')
plt.plot(epochs, acc, 'red', label='Training acc')
plt.plot(epochs, val_acc, 'blue', label='Validation acc')
plt.legend()

plt.figure()
plt.title('Training and validation loss')
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')

plt.legend()

plt.show()
```



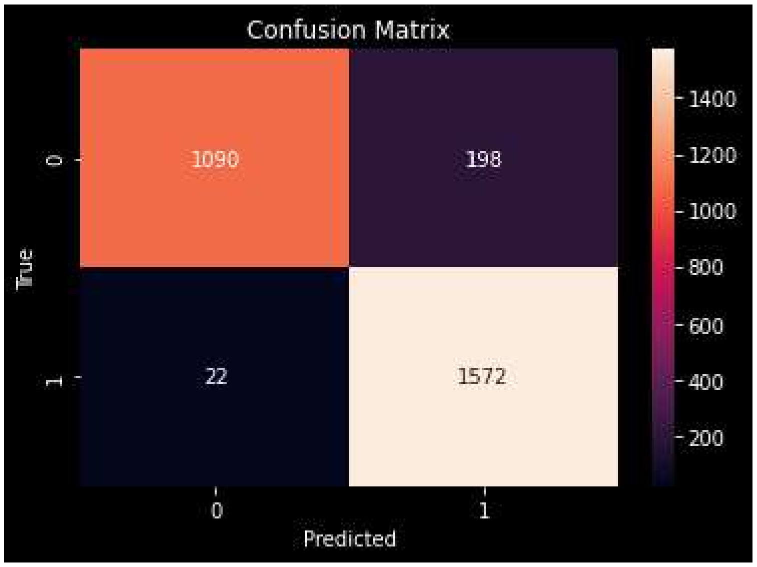
```
score_CNN = CNN.evaluate(test_set)
print("Test Loss:", score_CNN[0])
print("Test Accuracy:", score_CNN[1])
```

```
91/91 [==============================] - 585s 6s/step - loss: 0.1796 - accuracy: 0.9237
Test Loss: 0.17964863777160645
Test Accuracy: 0.9236640930175781
```

```
y_pred_CNN = CNN.predict(test_set)
y_pred_CNN = np.round(y_pred_CNN)

recall_CNN = recall_score(y_test, y_pred_CNN)
precision_CNN = precision_score(y_test, y_pred_CNN)
f1_CNN = f1_score(y_test, y_pred_CNN)
roc_CNN = roc_auc_score(y_test, y_pred_CNN)


print(classification_report(y_test, y_pred_CNN))
```

```
              precision    recall  f1-score   support

           0       0.98      0.85      0.91      1288
           1       0.89      0.99      0.93      1594

    accuracy                           0.92      2882
   macro avg       0.93      0.92      0.92      2882
weighted avg       0.93      0.92      0.92      2882
```

```
plt.figure(figsize = (6, 4))

sns.heatmap(confusion_matrix(y_test, y_pred_CNN),annot = True, fmt = 'd')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")

plt.show()
```



```
# Save the model
modelFileName = 'male-female-classifier.h5'
CNN.save(modelFileName)
print('model saved as', modelFileName)
```

```
    model saved as male-female-classifier.h5
```

```
CNN_base_inc = InceptionV3(input_shape = (75, 75, 3), include_top = False, weights = 'imagene
for layer in CNN_base_inc.layers:
    layer.trainable = False
```

```
    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/incep
    87916544/87910968 [==============================] - 1s 0us/step
```

```
x = layers.Flatten()(CNN_base_inc.output)
```

```
# Compilation
CNN_inc.compile(optimizer = RMSprop(lr = 0.0001), loss = 'binary_crossentropy', metrics = ['a

# Start of counting time
start = dt.datetime.now()

# Training and validation
CNN_inc_history = CNN_inc.fit(training_set, epochs = 50, validation_data = validation_set, ca

# End of Time Counting
end = dt.datetime.now()
time_CNN_inc = end - start
print ('\nTraining and validation time is: ', time_CNN_inc)
```

```
acc = CNN_inc_history.history['accuracy']
val_acc = CNN_inc_history.history['val_accuracy']
loss = CNN_inc_history.history['loss']
loss = CNN_inc_history.history['loss']
val_loss = CNN_inc_history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.title('Training and validation accuracy')
plt.plot(epochs, acc, 'red', label='Training acc')
plt.plot(epochs, val_acc, 'blue', label='Validation acc')
plt.legend()

plt.figure()
plt.title('Training and validation loss')
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')

plt.legend()

plt.show()


score_inc = CNN_inc.evaluate(test_set)
print("Test Loss:", score_inc[0])
print("Test Accuracy:", score_inc[1])


y_pred_inc = CNN_inc.predict(test_set)
y_pred_inc = np.round(y_pred_inc)

recall_inc = recall_score(y_test, y_pred_inc)
precision_inc = precision_score(y_test, y_pred_inc)
f1_inc = f1_score(y_test, y_pred_inc)
roc_inc = roc_auc_score(y_test, y_pred_inc)


print(classification_report(y_test, y_pred_inc))


plt.figure(figsize = (6, 4))

sns.heatmap(confusion_matrix(y_test, y_pred_inc),annot = True, fmt = 'd')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")

plt.show()


# Save the model
modelFileName = 'fire_classifier_model-inc.h5'
CNN_inc.save(modelFileName)
print('model saved as', modelFileName)
```

1s completed at 10:53 PM