False Face Detection with Multispectral NIR Image
Udacity Machine Learning Nanodegree Capstone

Eduardo Carvalho Nunes

March 16, 2020

1 Definition

1.1 Project Overview

Biometric System with Facial Recognition is an active topic of research in the last decades [1]. Currently, organizations are planning or using facial recognition systems, some examples: the Chinese government that implemented a facial recognition system on surveillance cameras around the city in Xinjiang [2]; in London at Gatwick airport, it will be the first airport in the United Kingdom to use facial recognition cameras for passenger identification [4] and in Brazil had his first arrest with the help of facial recognition [3].

However, some facial recognition systems may fail to detect fake faces. A test carried out by Consumentenbond (a Dutch non-profit organization that promotes consumer protection) in April 2019 found that of 110 mobile phones, 42 failed to detect false faces. The researchers used photographs printed face, masks and 3D heads for testing [5].

Control systems that use face recognition need more reliability. A system capable of performing this task automatically and correctly brings a series of practical advantages in the field of biometric authentication. So, this work focuses on using machine learning models to detect fake faces in the authentication of a facial biometry system.

Figure 1 illustrates the proposed architecture of a facial recognition system to be installed in a time and attendance control system. The system is summarized in a client-server model, where the client is a Raspberry Pi that sends the information and a user's face image in base64 format for authentication. The server receives the information and the face image. It then decodes the base64 format and processes it for authentication and recognition. After processing, the server will return an authentication result to the client.

1.2 Motivation and Problem Statement

To control the presence of students enrolled in the disciplines, the university where I study uses a presence control system. This system integrates presence terminals in the class register, where the student authenticates from an identification card by Radio Frequency. However, this system can be easily defrauded by a person who has user cards registered in the system. For example, a student who has other students' cards in hand, can make presence records in their absence. Thus, the main motivation of this project is to implement learning models to detect fake faces.

The general objective of this project is to use images of fake faces and real faces obtained by an infrared camera to detect fraud. For to detect fake face was implement the classifiers Decision Tree, Random Forest, KNN (K-closest neighbors) and SVM (Support Vector Ma-chine).

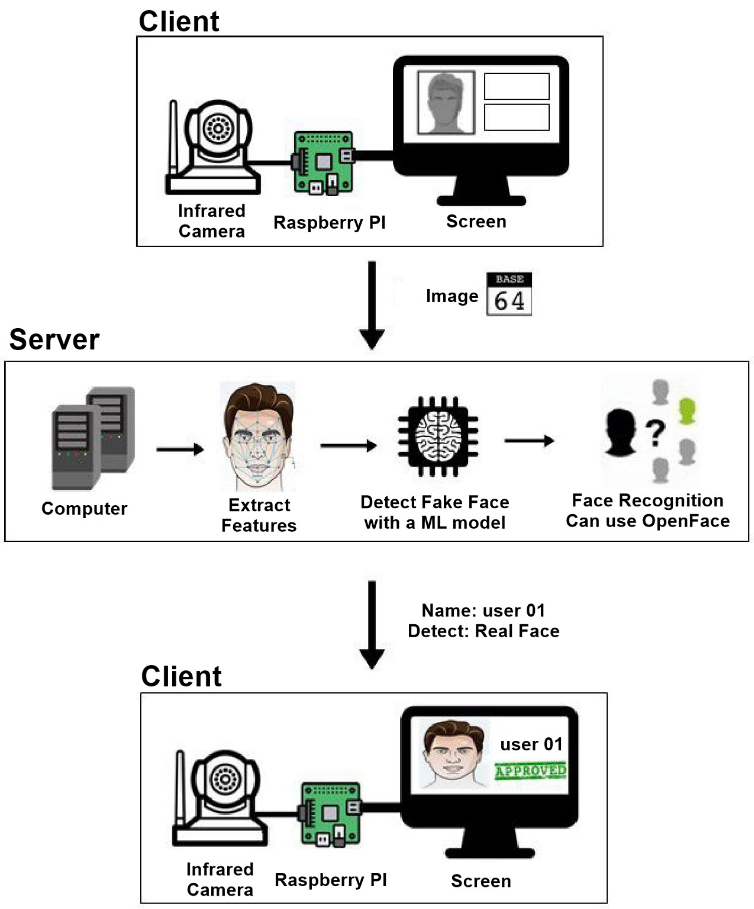
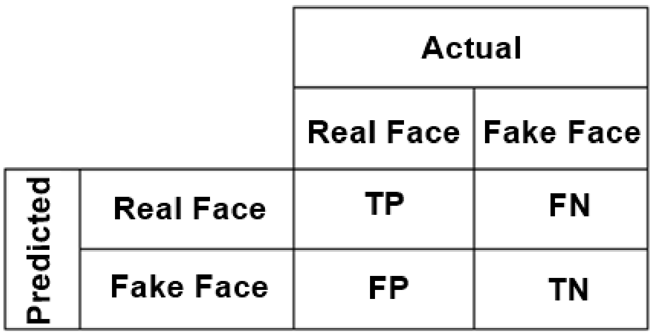


Figure 1: Proposed Architecture with Detect Fake

1.3 Metrics

To visualize (Figure 2) the performance of a classifier, a confusion matrix can be used. This matrix illustrates the number of correct (rows) and incorrect (columns) predictions in each class. For greater understanding, the binary classification between the fake face and real face is imagined, so the four measures of the confusion matrix will be:



		Actual	
		Real Face	Fake Face
Predicted	Real Face	TP	FN
	Fake Face	FP	TN

Figure 2: Confusion Matrix

- **True Positive (TP)**: true positive is the face that is real and that was correctly classified, real face;
- **True Negative (TN)**: True negative is the face that is false and that was correctly classified, the fake face;
- **False Positive (FP)**: false positive is the face that is false and that was mistakenly classified as the real face;

- **False Negative (FN)**: false negative is the face that is real and that was mistakenly classified as a fake face.

1.3.1 Metrics for Classification

From the confusion matrix, it is possible to evaluate the performance of the model in some metrics. The measurements are calculated from the metrics:

- **Accuracy**: calculated by adding the diagonal values of the matrix and dividing by the sum of the values of all elements of the matrix (Equation 1).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

- **Recall or Sensitivity**: corresponds to the hit rate of the positive class (Equation 2).

$$Sensitivity = \frac{VP}{VP + FN} \tag{2}$$

- **Specificity**: corresponds to the hit rate in the negative class (Equation 3).

$$Specificity = \frac{VN}{VN + FP} \tag{3}$$

2 Analysis

2.1 Data Exploration and Visualization

2.1.1 Dataset

The dataset was built by the author himself. So far, the data set has 15 participants (my friends), between 22 and 30 years old. Each participant has 10 images of real faces and 20 images of fake faces (10 fake digital images and 10 fake images on paper). In total, the data set contains 450 images.The Figure 3 shows an example of a participant where we have 10 images of real faces and 20 fake faces.

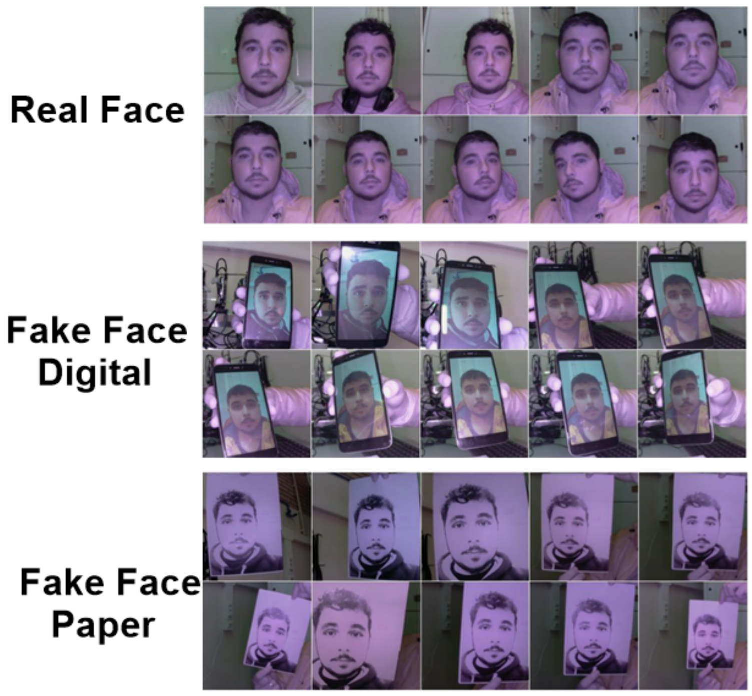


Figure 3: Images of real face and fake face of a participant

2.1.2 Feature Extraction

It was use the OpenFace library to extract features. According to the website [?], OpenFace is a facial recognition library with *deep learning*.

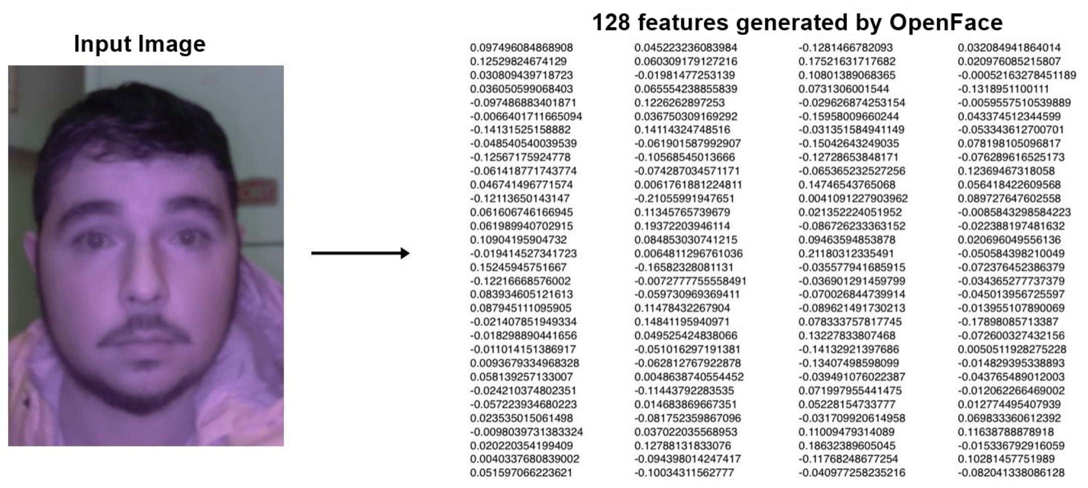
The OpenFace procedure for single input image are:

- Detect face with pre-trained dlib or OpenCV models;
- Cut out the detected face and use it as an entry in the deep neural network;
- Use a *deep learning* to represent the face in a unitary hypersphere of 128 dimensions (characteristics);
- After acquiring the 128 characteristics, apply *cluster* or classification techniques to complete the facial recognition task.

Figure 5 below shows an example using OpenFace, where the input is an image with a face and the output is a vector with 128 values that represent the characteristics of the face.

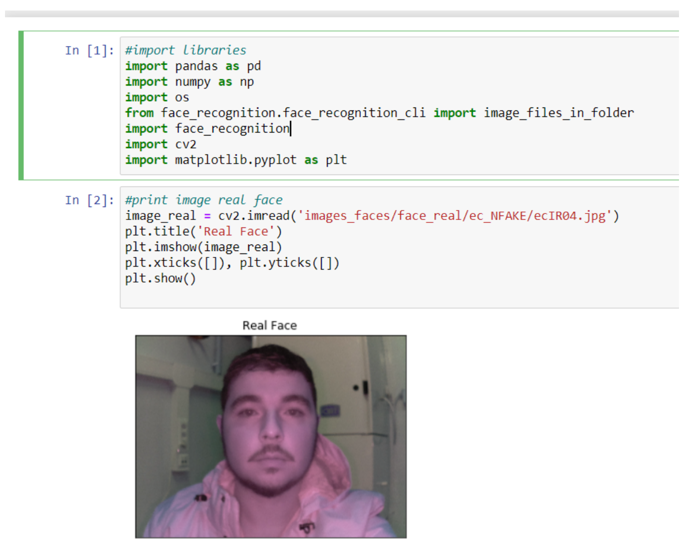
2.1.3 Image Sample File Data Overview

These images are files .jpg format. For image analysis, it was used the following main libraries: Face recognition (contains the dlib and openface libraries);OpenCV; Numpy and Pandas.


Input Image
128 features generated by OpenFace

0.097496084868908	0.045223236083984	-0.1281466782093	0.032084941864014
0.12529824674129	0.060309179127216	0.17521631717682	0.020976085215807
0.030809439718723	-0.01981477253139	0.10801389068365	-0.00052163278451189
0.036050599068403	0.065554238855839	0.0731306001544	-0.1318951100111
-0.097486883401871	0.1226262897253	-0.029626874253154	-0.0059557510539889
-0.0066401711665094	0.036750309169292	-0.15958009660244	0.043374512344599
-0.14131525158882	0.14114324748516	-0.031351584941149	-0.053343612700701
-0.048540540039539	-0.061901587992907	-0.15042643249035	0.078198105096817
-0.12567175924778	-0.10568545013666	-0.12728653848171	-0.076289616525173
-0.061418771743774	-0.074287034571171	-0.065365232527256	0.12369467318058
0.046741496771574	0.0061761881224811	0.14746543765068	0.056418422609568
-0.12113650143147	-0.21055991947651	0.0041091227903962	0.089727647602558
0.061606746166945	0.11345765739679	0.021352224051952	-0.0085843298584223
0.061989940702915	0.19372203946114	-0.086726233363152	-0.022388197481632
0.10904195904732	0.084853030741215	0.09463594853878	0.020696049556136
-0.019414527341723	0.0064811296761036	0.21180312335491	-0.050584398210049
0.15245945751667	-0.16582328081131	-0.035577941685915	-0.072376452386379
-0.12216668576002	-0.0072777755558491	-0.036901291459799	-0.034365277737379
0.083934605121613	-0.059730969369411	-0.070026844739914	-0.045013956725597
0.087945111095905	0.11478432267904	-0.089621491730213	-0.013955107890069
-0.021407851949334	0.14841195940971	0.078333757817745	-0.17898085713387
-0.018298890441656	0.049525424838066	0.13227833807468	-0.072600327432156
-0.011014151386917	-0.051016297191381	-0.14132921397686	0.0050511928275228
0.0093679334968328	-0.062812767922878	-0.13407498598099	-0.014829395338893
0.058139257133007	0.0048638740554452	-0.039491076022387	-0.043765489012003
-0.024210374802351	-0.11443792283535	0.071997955441475	-0.012062266469002
-0.057223934680223	0.014683869667351	0.05228154733777	0.012774495407939
0.023535015061498	-0.081752359867096	-0.031709920614958	0.069833360612392
-0.0098039731383324	0.037022035568953	0.11009479314089	0.11638788878918
0.020220354199409	0.12788131833076	0.18632389605045	-0.015336792916059
0.0040337680839002	-0.094398014247417	-0.11768248677254	0.10281457751989
0.051597066223621	-0.10034311562777	-0.040977258235216	-0.082041338086128

Figure 4: Facial features of a real face using OpenFace



```
In [1]: #import libraries
        import pandas as pd
        import numpy as np
        import os
        from face_recognition.face_recognition_cli import image_files_in_folder
        import face_recognition
        import cv2
        import matplotlib.pyplot as plt
```

```
In [2]: #print image real face
        image_real = cv2.imread('images_faces/face_real/ec_NFAKE/ecIR04.jpg')
        plt.title('Real Face')
        plt.imshow(image_real)
        plt.xticks([]), plt.yticks([])
        plt.show()
```

Figure 5: Example for extract features using OpenFace

2.2 Algorithms and Techniques

2.2.1 Decision Tree

Learning by Decision Tree is a method that allows the modeling of discrete systems in which the model obtained is a tree. Decision Tree is one of the most used methods for inductive inference because it is robust to noise in the data, allowing a graphic representation of the generated model[7].

The representation of the Decision Tree consists of a structure in which each internal node (not a leaf) corresponds to an attribute to be tested, each descending branch represents a possibility for this test and each leaf contains the class corresponding to the instances classified by it where it is. decision obtained after testing the attributes sequentially. The path taken to reach the class corresponds to a classification rule[7].

An example of Decision Tree is shown in the Figure 6, the decision on which activity (leaf) to do on the weekend, may depend on whether you have it alone or with friends (root node), in both cases, the decision also depends on the climate (children nodes). If it is sunny and friends are available, you can play soccer, if it rains, you can go to the cinema. If you are alone, no matter the time, you will play video games.

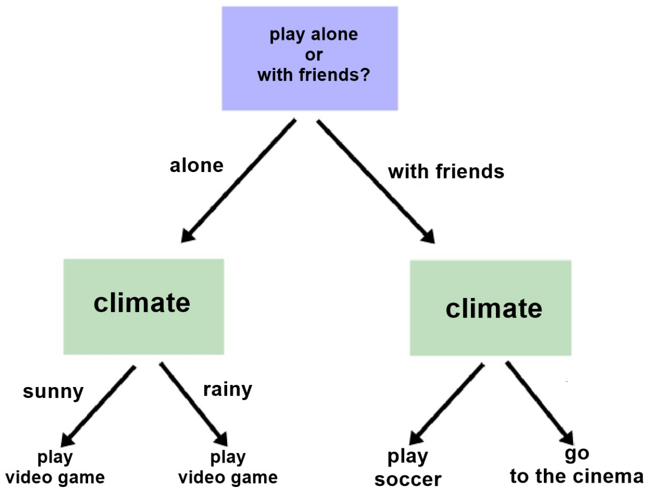


Figure 6: Example Decision Tree

2.2.2 Random Forest

Random Forest (RF) is a supervised learning classifier that evolves from decision trees. Smith and Koning [7] define RF as an algorithm that uses multiple decision trees to predict an outcome, and this collection of trees is often called a set. Each individual RF tree has a class forecast and the class with the most votes becomes the model forecast.

The Figure 7 illustrates an example of RF. In the RF it contains six trees and each tree makes a prediction of class 1 or class 0. The final result of the prediction was class 1, because it had more votes (4 votes).

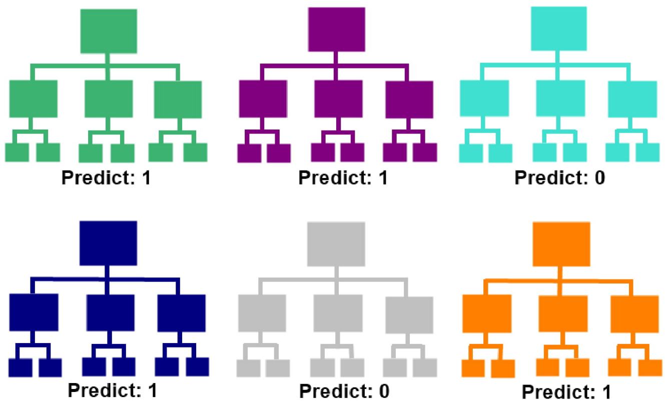


Figure 7: Example Random Forest

2.2.3 KNN

K-Nearest Neighbors (KNN) is a supervised machine learning classifier that uses instance-based learning techniques, where the distance of a new example (new instance) is calculated from each instance belonging to the knowledge base [8]. In other words, KNN assumes that similar things are close to each other.

The KNN classifier has two important characteristics that precede the prediction of a new instance. The first is the definition of the k number of closest neighbors that will compose the neighborhood of the new instance. The second is the choice of the distance measure responsible for identifying the k observations of the set of instances closest to the new instance [9].

The Figure 8 illustrates a simple example of KNN. Note that in the center of the figure there is an instance (square) that is not classified and all other instances are classified (blue and red), each with its class (star and circle). To classify the new instance (square), the distance with all instances must be calculated, to identify the shortest distances. For k = 3, check the three closest neighbors and the new instance will be classified as a red circle. For k = 5, check the five closest neighbors and the new instance will be classified as a blue star.

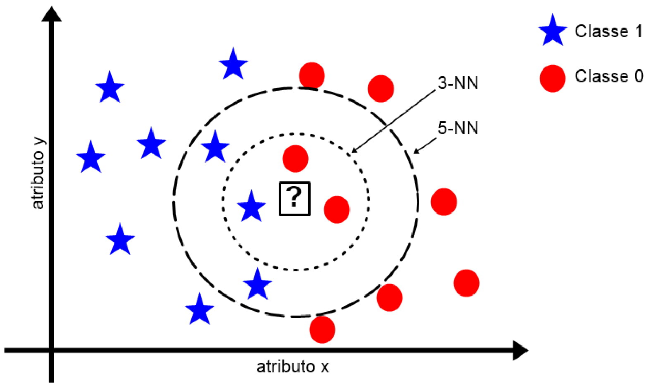


Figure 8: Example KNN. Class 0 are represented with red circles and Class1 are represented with blue stars

2.3 Benchmark Model

In this section, analyzes of published works that refer to fake face detection will be described. A brief search in the literature, there are numerous works on the detection of false faces using only visible camera or RGB. However, only two studies were found related to the detection of fake faces using infrared (IR) camera and machine learning. Table 1 shows the related works in summary.

Table 1: Title, year, sensor, machine learning model, dataset e results.

Title	**Year**	**Sensor**	**Model**	**Dataset**	**Performance**
Reliable Face Anti-Spoofing Using Multispectral SWIR Imaging [10]	2016	IR	SVM	404; images; 137 participants	Accuracy:99:28%
A Dataset and Benchmark for Large-scale Multi-modal Face Anti-spoofing[11]	2019	RGB, IR e Depth	CNN	5.175.790; images; 21000 videos; 1000 participants	TPR(RGB):49.3% TPR(IR):65.3% TPR(Depth):88.3% TPR(RGB+IR+Depth): 96.7%

The paper by Steiner et al. [10] published in June 2018, uses an infrared camera in the SWIR spectrum and the SVM model of machine learning to detect false faces. The dataset used contains 404 images from 137 participants. The accuracy result for fake face detection was 99.28 %.

The paper by Zhang et al. [11] published in April 2019 used the Intel RealSense SR300 camera that contains IR, RGB and Depth to capture faces. The paper features the largest fake-face *dataset* in literature with more than 5 million images and 21,000 videos from 1000 participants. For fake face detection, the authors used the CNN deep learning model. As the camera contains 3 sensors, faces were analyzed only with the normal camera (RGB), IR camera, a depth camera and with 3 sensors together. Using only the RGB camera obtained the TPR (Sensitivity) of 49.3 %, with IR camera the TPR was 65.3 %, with the depth camera the TPR was 88.3 % and finally, using the 3 sensors the TPR was 96.7 %.

3 Methodology

3.1 Data Processing and Data Splitting

3.1.1 Extract Features - Function

The fake and real faces are separated by folders. The function was created (getFeaturesOpenFace) to extract features. The function receives the path of the folder with the images and returns a dictionary with N instances and 128 columns (features).

```
import numpy as np

def getFeaturesOpenFace(image_folder):
    """Return

    Parameters:
        image_folder (string): Path folder with faces images

    Returns:
        dict_features_openface (dictionary): Dictionary where each lane has a float

    """

    dict_features_openface = {}

    for person in os.listdir(image_folder):
        if not os.path.isdir(os.path.join(image_folder, person)):
            continue
        cont = 1
        name = str(person)
        # Loop through each training image for the current person
        for img_path in image_files_in_folder(os.path.join(image_folder, person)):
            print(img_path)
            image = face_recognition.load_image_file(img_path)
            #features
            features=face_recognition.face_encodings(image)[0]
            dict_features_openface[name + "_" + str(cont)] = features
            cont = cont +1

    return dict_features_openface
```

Example call function:

```
#get features of real faces
dict_features_real = getFeaturesOpenFace("images_faces/face_real")
```

3.1.2 Dictionary to csv format

A function was created to convert a dictionary to csv.

```
def dict_to_csv(dict_features, nameFile):
    """Return

    Parameters:
        dict_features (dictionary): Path folder with faces images
        nameFile (String) : name of .csv

    Returns:
        dict_features_openface (dictionary): Dictionary where each lane has a float
    """
    aux = pd.DataFrame.from_dict(dict_features, orient='index')
    export_csv = aux.to_csv (r'datas_csv/dataset-' + nameFile + '.csv', header=True)

    return export_csv
```

3.1.3 Label

The classification of this dataset is binary (fake face or real face). A value of 0 for fake and 1 for real face was assigned.

After extracting the features and assigning labels, a dataset-all was built containing all faces.

```
#read datasets csv with pandas
df_face_real = pd.read_csv('datas_csv/dataset-real-faces.csv', delimiter = ',')
df_fake_paper = pd.read_csv('datas_csv/dataset-fake-faces-paper.csv', delimiter = ',
df_fake_digital = pd.read_csv('datas_csv/dataset-fake-faces-digital.csv', delimiter

df_face_real = df_face_real.drop(columns=['Unnamed:␣0'])
df_fake_paper = df_fake_paper.drop(columns=['Unnamed:␣0'])
df_fake_digital = df_fake_digital.drop(columns=['Unnamed:␣0'])

#add column LABEL where 0 is FAKE and 1 is NOT FAKE
column_fake_paper = [0] * (len(df_fake_paper))
column_fake_digital = [0] * (len(df_fake_digital))
column_real = [1] * (len(df_face_real))

#insert column 1  position
df_fake_paper.insert(loc=0, column='label', value=column_fake_paper)
df_fake_digital.insert(loc=0, column='label', value=column_fake_digital)
df_face_real.insert(loc=0, column='label', value=column_real)

#dataset with all faces (fake and real)
df_dataset_all = df_face_real.append([df_fake_paper, df_fake_digital])
df_dataset_all

#dataframe to csv
df_dataset_all.to_csv (r'datas_csv/dataset-all.csv', index = False, header=True)
```

3.1.4 Split Dataset

It was split the dataset into training and test sets. The testing set size was 30% and it was set a random state.

```
# read dataset ALL from csv
df = pd.read_csv("datas_csv/dataset-all.csv")
#get FEATURES
x = df.drop(columns=['label'])
# get LABEL
y = df['label']
# 70% for TRAIN and 30% for TEST
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

#save dataframe to csv
X_train.to_csv(r'train/X_train.csv', index = False, header=True)
```

```
X_test.to_csv(r'test/X_test.csv', index = False, header=True)
y_train.to_csv(r'train/y_train.csv', index = False, header=True)
y_test.to_csv(r'test/y_test.csv', index = False, header=True)
```

3.2 Implementation

3.2.1 Parameters of Machine Learning models

A grid search was performed to obtain models with the best parameters. The parameters of the following classifiers (Table 2).

Table 2: Parameters of machine learning models

Classifier	**Parameters**
Decision Tree	depth=2,3,4,5,6,7, ..., 32

Random Forest	n estimators = [200, 500]
	max features = auto, sqrt and log2
	maxdepth = 4,5,6,7 and 8
	criterion = gini and entropy

KNN	K=3,5,7,9,11,15 and 19
	weights = uniform and distance
	metric = euclidean and manhattan

SVM	kernel=RBF,
	C=0.1,1,10 and 100
	gamma=0.001, 0.01, 0.1, and 1

4 Results

4.1 Decision Tree - Fake Fake x Real Face Detection

The parameters of the decision tree model were: max depth =11. See the code below:

```
#Decision tree Classifier
dtc = DecisionTreeClassifier(max_depth=11)
dtc = dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

print('accuracy: ' + str(accuracy_score(y_test, y_pred)))

print(classification_report(y_test, y_pred))

accuracy: 0.9271523178807947
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       108
           1       0.92      0.81      0.86        43

    accuracy                           0.93       151
   macro avg       0.93      0.89      0.91       151
weighted avg       0.93      0.93      0.93       151

Confusion matrix, without normalization
[[105   3]
 [  8  35]]
```

Table 3 shows the confusion matrix for this model. For 108 fake faces the model classified 105 correctly or 97% of sensitivity. For 43 real faces it classified 35 correctly or 81% specificity.

Table 3: Confusion Matrix - Decision Tree

		Prediction	
	total:151	Fake Face	Real Face
Real	Fake Face	105	3
	Real Face	8	35

4.2 KNN - Fake Fake x Real Face Detection

The parameters of the decision tree model were: metric=euclidean, n neighbors=3 and weights=distance. See the code below:

```
#KNN Classifier
knn = KNeighborsClassifier(metric= 'euclidean', n_neighbors= 3, weights= 'distance')
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

print('accuracy: ' + str(accuracy_score(y_test, y_pred)))

print(classification_report(y_test, y_pred))

accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       108
           1       1.00      1.00      1.00        43

    accuracy                           1.00       151
   macro avg       1.00      1.00      1.00       151
weighted avg       1.00      1.00      1.00       151

Confusion matrix, without normalization
[[108   0]
 [  0  43]]
```

Table 6 shows the confusion matrix for this model. For 108 fake faces the model classified 108 correctly or 100% of sensitivity. For 43 real faces it classified 43 correctly or 100% specificity.

Table 4: Confusion Matrix - KNN

		Prediction	
	total:151	Fake Face	Real Face
Real	Fake Face	108	0
	Real Face	0	43

4.3 Random Forest - Fake Fake x Real Face Detection

The parameters of the decision tree model were: random state=42, max features=auto, n estimators=500 and criterion=500. See the code below:

```
#Random Forest Classifier
rfc = RandomForestClassifier(random_state=42, max_features='auto', n_estimators= 500
criterion='entropy')

rfc.fit(X_train, y_train.values.ravel())

y_pred = rfc.predict(X_test)

import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

print('accuracy:␣' + str(accuracy_score(y_test, y_pred)))

print(classification_report(y_test, y_pred))

accuracy: 0.9867549668874173
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       108
           1       1.00      0.95      0.98        43

    accuracy                           0.99       151
   macro avg       0.99      0.98      0.98       151
weighted avg       0.99      0.99      0.99       151

Confusion matrix, without normalization
[[108   0]
 [  2  41]]
```

Table 6 shows the confusion matrix for this model. For 108 fake faces the model classified 108 correctly or 100% of sensitivity. For 43 real faces it classified 41 correctly or 95% specificity.

Table 5: Confusion Matrix - Random Forest

		Prediction	
	total:151	Fake Face	Real Face
Real	Fake Face	108	0
	Real Face	2	41

4.4 SVM - Fake Fake x Real Face Detection

The parameters of the decision tree model were: C=10, gamma=1 and kernel=rbf. See the code below:

```
#SVM Classifier
svm = SVC(C=10, gamma=1, kernel='rbf')
svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)

import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

print('accuracy:␣' + str(accuracy_score(y_test, y_pred)))

print(classification_report(y_test, y_pred))

accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       108
           1       1.00      1.00      1.00        43

    accuracy                           1.00       151
   macro avg       1.00      1.00      1.00       151
weighted avg       1.00      1.00      1.00       151

Confusion matrix, without normalization
[[108   0]
 [  0  43]]
```

Table 6 shows the confusion matrix for this model. For 108 fake faces the model classified 108 correctly or 100% of sensitivity. For 43 real faces it classified 43 correctly or 100% specificity.

Table 6: Confusion Matrix - SVM

		Prediction	
	total:151	Fake Face	Real Face
Real	Fake Face	108	0
	Real Face	0	43

4.5 Comparison of Machine Learning Models for Detecting Fake Faces

In this section, the results of the classifications of machine learning models will be compared. Table 7 show the percentages of accuracy.

Looking at the Table 7 we can see that the best results were the KNN and SVM models. Random Forest also had good results. The worst of all was the Decision Tree.

Table 7: Results of Accuracy, Sensitivity and Specificity

Classifier	**Accuracy**	**Sensitivity**	**Specificity**
Decision Tree	93%	97%	81%
KNN	100%	100%	100%
Random Forest	99%	100%	95%
SVM	100%	100%	100%

4.6 Conclusions

This project shows a study for detecting fake faces using machine learning, where it can be used as a pre-processing for a facial recognition system.

A data set was created for training and testing. For the creation of a camera data set with NIR and Raspberry Pi 3 infrared light, it proved to be efficient for capturing images of fake faces and real faces.

Four machine learning classifiers were implemented to detect fake faces: Decision Tree, Random Forest, KNN and SVM. The model that presented the best result was SVM and KNN with 100% accuracy.

These results are satisfactory if compared with the work of Steiner [10] with 99.28% accuracy and Zhang [11] with 65.3% (infrared light) and 96.7% (infrared light, depth and light visible).

In general, it was concluded that the use of a NIR infrared camera with machine learning techniques demonstrated that it is possible to create a fake face detector, which can be performed automatically in a facial recognition system.

4.7 Future Works

As future work, it is suggested to increase the database with new participants, expanding the diversity of the dataset. With the increase of the dataset, new training and tests of the machine learning models are suggested for the detection of fakeface and real face. It is also suggested to create false faces like 3D masks.

References

[1] Jain, Anil K., and Stan Z. Li. Handbook of face recognition. Vol. 1. New York: springer, 2011.

[2] China's massive investment in artificial intelligence has an insidious downside. (2020). Retrieved 17 February 2020, from https://www.sciencemag.org/news/2018/02/china-s-massive-investment-artificial-intelligence-has-insidious-downside

[3] Carnival has first arrested via facial recognition camera in Brazil. (2020). Retrieved 17 February 2020, from https://www.tecmundo.com.br/seguranca/139262-carnaval-tem-primeiro-preso-via-camera-reconhecimento-facial-brasil.htm

[4] Gatwick Airport commits to facial recognition tech at boarding. (2020). Retrieved 17 February 2020, from https://www.bbc.com/news/technology-49728301

[5] Kulche, P. (2020). Retrieved 10 March 2020, from https://www.consumentenbond.nl/veilig-internetten/gezichtsherkenning-te-hacken

[6] Sunasra, M. (2020). Performance Metrics for Classification problems in Machine Learning. Retrieved 11 March 2020, from https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b

[7] C. Smith,Decision trees and random forests: a visual introduction for beginners.Blue Windmill Media, 2017.

[8] . M. Mitchell,Machine Learning, 1aed. New York, NY, USA: McGraw-Hill, Inc.,1997,isbn: 0070428077, 9780070428072.

[9] S. Raschka e V. Mirjalili,Python machine learning. Packt Publishing Ltd, 2017.

[10] H. Steiner, A. Kolb e N. Jung, "Reliable face anti-spoofing using multispectral swirimaging", em2016 International Conference on Biometrics (ICB), IEEE, 2016,pp. 1–8.

[11] S. Zhang, X. Wang, A. Liu, C. Zhao, J. Wan, S. Escalera, H. Shi, Z. Wang e S. Z. Li,"A Dataset and Benchmark for Large-scale Multi-modal Face Anti-spoofing", emProceedings of the IEEE Conference on Computer Vision and Pattern Recognition,2019, pp. 919–928.