Al-Imam Muhammad Bin Saud Islamic University

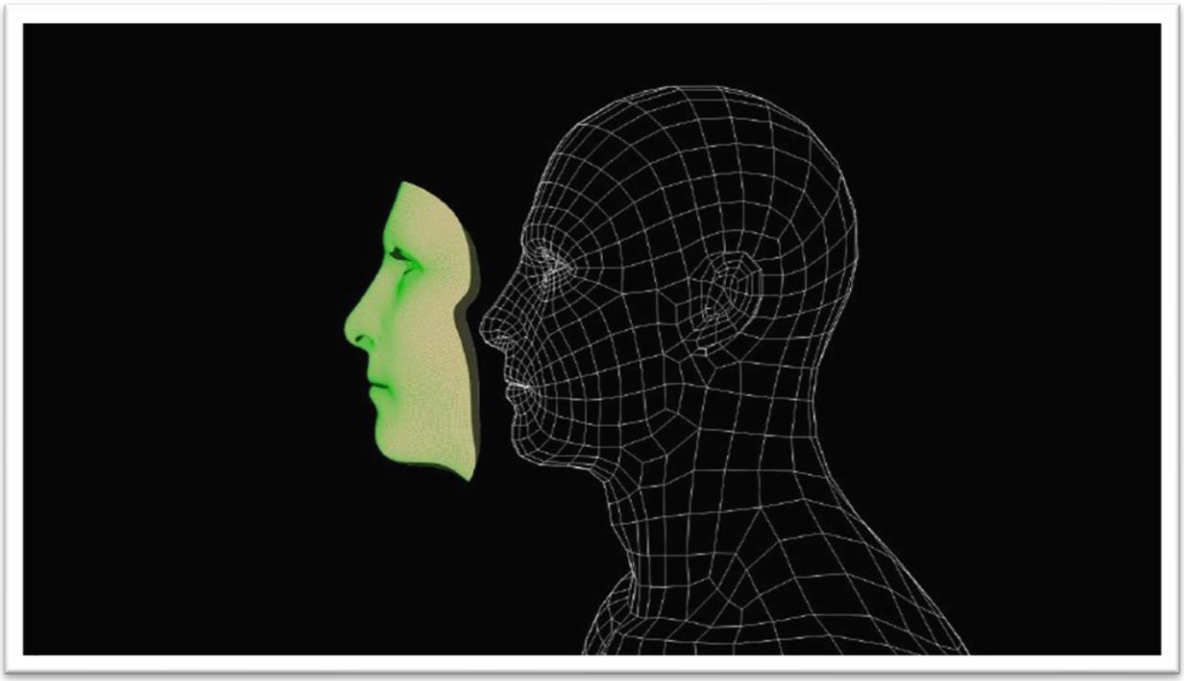
College of Computer and Information Sciences,

Department of Computer Science

CS 332 Information Security

Course project

May 14th, 2022



Student Name	*Student ID*	*Section*
Khloud Alnufaie	440020617	371
Alhanouf Almansour	440019183	371
Raghad Albosais	440020209	371
Weaam Alghaith	440023306	371

Content

1.Introduction

List of Figures

1. Introduction

In light of our accelerating world and the huge amount of data transmitted over the internet specifically in social media, an individual sees dozen or even hundreds of images and types of media every day. Deepfake is a newly emerged issue in our modern days which is media of a person in which their face or body has been digitally altered so that they appear to be someone else, typically used maliciously or to spread false information. In this project, we aim to detect deepfake images using machine learning techniques, which supports vector machine SVM. We have used transfer learning strategy to extract features from images, through using pre-trained CNN model, which is VGG16 combined with SVM classifier that is trained using the extracted features provided by VGG16. To illustrate the functionality in the model, we have included production by deployment model.

1.1 Problem Statement

One of the strong techniques used in creating misinformation has become known recently as "Deepfake". Deepfakes increasingly threaten the privacy of individuals. Furthermore, Deepfakes can distort our perception of the truth and deceive us. The content of an image can shake the world either because it sparks controversy, or discredits someone. An individual may be accused or suspected of a situation that did not actually occur. For example, modifying a person's expression to appear sad when in reality, they were happy to satisfy a fake narrative.

1.2 Problem formulation

Task (T)	Classify an image to a real or a fake depend on it is manipulated or not.
Experience (E)	A collection of real and fake faces images
Performance (P)	Classification evaluation metric, which is precision. It is the number of images predicted correctly true positives (TP) over the number of true positives plus the number image predicted incorrectly of false negatives (FN)

1.3 Model architecture

Here we illustrated the model architecture that we followed to apply a solution for Deepfake detection of images. The image is preprocessed using three image preprocessing techniques (image resize, color space conversion, and normalization). Next, the image is moved into a pretrained convolutional neural network (CNN) feature extractor, called VGG16 which extracting features from the image. After that, the output of the feature extractor is fed as input to the support vector machine (SVM) classifier.

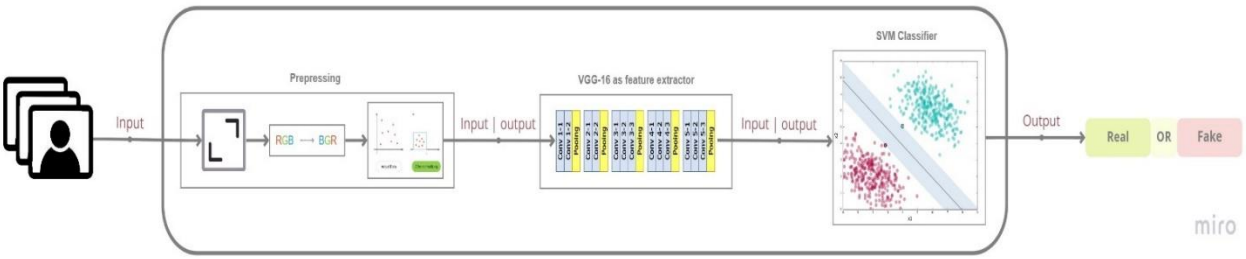


Figure 1. Overall process of our project

2. Dataset overview

We plan to detect fake image by using "Real and Fake Face Detection" challenge dataset of Kaggle. The dataset contains expert-generated high-quality photoshopped face images. The images are composite of different faces, separated by eyes, nose, mouth, or whole face. The full dataset contains 452 MB of faces Inside the parent directory, training_real/training_fake contains real/fake face photos, respectively. In case of fake photos, they have three groups: easy, mid, and hard. We plan to use 500 images with ground truth, split them as 75% training, and 25% test to evaluate models using this.

Dataset available: https://www.kaggle.com/datasets/ciplab/real-and-fake-face-detection.

3. Step of implementation

3.1 Import resources

- Import resources we need in implementation

```
# install free open source Python library (keras) for developing and evaluating deep learning models.
!pip install keras
```

```
#NumPy :is a Python library used for convert list to arrays
import numpy as np

#plt :module to import name for opencv-python  library to solve computer vision problems.
import matplotlib.pyplot as plt

#glob :module used to return all file paths that match in folders
import glob

#module :to import name for opencv-python  library to solve computer vision problems
import cv2

 #pd :package used for data analysis and manipulation tool
import pandas as pd

#BatchNormalization :is a technique used during training
from tensorflow.keras.layers import BatchNormalization

#os :module to provides functions for dealing with folder
import os
```

```
#sns :library for data visualization
import seaborn as sns

#VGG16 :is CNN tecnique used for feature extractor
from keras.applications.vgg16 import VGG16

#SVC (support vector classifier) :used to classification the images
from sklearn.svm import SVC

#GridSearchCV :is a library to Build and Train SVC with Hyperparameter
from sklearn.model_selection import GridSearchCV

#make_scorer :function for take score of GridSearchCV
from sklearn.metrics import make_scorer

#precision_score : evaluation metric for the ratio tp / (tp + fp)
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score

#confusion_matrix,ConfusionMatrixDisplay : to dealing and show confusion_matrix
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay

#joblib :is tool used to save the model for deployment
import joblib
```

Figure 2. Import all required libraries

3.2 Preparing the data

In this step we install the dataset and explore the data and labels then we applied pre-processing to data to improve the quality of the image so that we can analyze it in a better way. In the last we split the data into two subsets.

3.2.1 Loading the data

- Load the data and get their paths

```
# load the paths of images
real = "/content/real_and_fake/real/"
fake = "/content/real_and_fake/fake/"

#to get the list of all real images
real_path = os.listdir(real)

#to get the list of all fake images
fake_path = os.listdir(fake)
```

Figure 3. Load data

- Create lists for data path and label

```
#declare a list for images path
image_path = []
#declare a list for images label (fake/real)
image_label = []

# loop for get the label (fake/real)
for directory_path in glob.glob("real_and_fake/*"):
    #loading the image label
    label = directory_path.split("/")[-1]
    # loop for load images path and label into list
    for img_path in glob.glob(os.path.join(directory_path, "*.jpg")):
        #add image path into image_path list
        image_path.append(img_path)
        #add image label into image_label list
        image_label.append(label)

# Build a dataframe for dataset
dataset = pd.DataFrame(data=list(zip(image_path, image_label)), columns=['imagename', 'class'], index=None)
```

Figure 4. Prepare two list for images paths and for image labels

3.2.2 Exploring the data

- Show some of data with its label

```
# Print the first 5 entries of the dataset
dataset.head()
```

Figure 5. Display data with its label

- Show the shape of data

```
# Number of rows(image) X columns (imagename,class)
dataset.shape
```

Figure 6. Display diminution of the data

- Show bar chart for data

```
# check the distribution of target varibale ( the data is balanced or not)
df = pd.DataFrame (image_label, columns = ['target'])
#create bar chart of the number of examples per classes
sns.countplot(x='target',data=df,palette=[[0, 0.75, 0.75],[0.4940, 0.1840, 0.5560] ])
#set the title
plt.xlabel('Deepfake dataset')
plt.title('')
#show the chart
plt.show()
```

Figure 7. Display dataset balancing of its classes

- Show some data with label

```
#loading the image
def load_img(path):
    #loads the image from its file
    image = cv2.imread(path)
    #check if the method returns an empty matrix.
    if image is None:
        print('Wrong path:', path)
    else:
        #determining the size of the image
        image = cv2.resize(image, (224, 224))
    #return the image
    return image[...,::-1]
#Visualization five sample of images
def display(end_path,start_path, Type):
    #resize the image
    fig = plt.figure(figsize=(10, 10))
    #loop to print fives images
    for i in range(5):
        #show the images in 4×4 display and incerment i
        plt.subplot(4, 4, i + 1)
        #call the load_img() to get image and show it
        plt.imshow(load_img(start_path + end_path[i]), cmap='gray')
        #the title of display
        plt.suptitle((Type +" faces"), fontsize=20)
        #set the axis propreties of graph
        plt.axis('off')

#To explore the dataset display sample of real image
display(real_path, real, 'Real')
#To explore the dataset display sample of fake image
display(fake_path, fake, 'Fake')
```

Figure 8. Display images with its label

3.2.3 Preprocess the data

- Preprocessing applied to data

```
# Resize images to 224
SIZE = 224

#Capture data and labels into respective lists
#declare a list for images
images = []
#declare a list for images label
labels = []

# loop for get the label (fake/real)
for directory_path in glob.glob("real_and_fake/*"):
    #loading the image label
    label = directory_path.split("/")[-1]
    #loop for preprocess the images
    for img_path in glob.glob(os.path.join(directory_path, "*.jpg")):
        #loads the image from its file and specifies to load a color image
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        #preprocessing : determining the size of the image
        img = cv2.resize(img, (SIZE, SIZE))
        #preprocessing : to convert an image from one color space to another
        #preprocessing : converting from RGB to BGR
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        # preprocessing: normalization
        img = img / 255.0
        #add image into images list
        images.append(img)
        #add image label into labels list
        labels.append(label)
```

Figure 9. Data preprocessing

- Encode labels from text to integers

```
#preprocessing : transformations applied to our data before feeding it to the model
from sklearn import preprocessing

#Encode labels from text to integers
#Encode class name to 1:real and 0:fake
le = preprocessing.LabelEncoder()

#Fit label encoder
le.fit(labels)

#Performs fit and transform on the labels at a single time
labels_encoded = le.transform(labels)
```

Figure 10. Convert text labels to integers labels

3.2.4 Splitting the data into train and test

- Splitting dataset into two subsets train set and test set

```
#train_test_split : splitting dataset into two subsets: for training data and for testing data.
from sklearn.model_selection import train_test_split

# X : images, y: labels
#Split data into test and train datasets
X_train, X_test, y_train, y_test = train_test_split(images, labels_encoded, train_size=0.75)

#give the number of row (sample) in train set and the size and number of colums
print(X_train.shape)

#give the number of row (sample) in test set and the size and number of colums
print(X_test.shape)
```

Figure 11. Data splitting

3.3 Model development

The workflow of our model is illustrated in Figure.1. Firstly, we load pre-trained convolutional neural network (CNN) feature extractor (VGG16) then, run the pre-trained model on our dataset to extract features from the image, after that the output of the feature extractor is feed as input to support vector machine (SVM) classifier. The pre-trained models are applied transfer learning concept, which taking the weight learned are trained(learned) on previously problem and used it on similar type of new problem without need to train the model again in the new problem. There are different types of pre-trained CNN models, the most used as feature extraction in deepfake detection task is the VGG16. The VGG16 model is a pre-trained CNN model on ImageNet dataset. It is used widely as a feature extractor, where the last layer (classification layer) is removed to give the classification task to another model. The benefit of pre-trained model is the weight that are learned by another similar task, so we do not need to again train the model (learn weights) in our task, we benefit from the weight that are learned previously, so we are freezing the layers to be non-trainable. The SVM is a supervised machine learning classifier. It is the best ML model used to detect deepfake.

3.3.1 Feature extraction using (VGG16)

- Load the VGG16 pre-trained model

```
# Load model wothout classifier/fully connected layers to be used as feature extractor
# to give the classification task to another model
VGG_model = VGG16(weights='imagenet', include_top=False, input_shape=(SIZE, SIZE, 3))

# Make loaded layers non-trainable. to use the pre-trained weights in our task
for layer in VGG_model.layers:
  layer.trainable = False
```

Figure 12. Step1 in Feature extraction

- Extract features using VGG16

```
# Use the pre-trained VGG16 to extract features from image (training dataset)
# the VGG16 expects color input images to be rescaled to the size of 224×224 squares
feature_extractor = VGG_model.predict(X_train)

# Change the shape of extracted features to be the required shape of input data to the SVM
# the required shape of SVM is 2D (number of rows, number of columns)
# the number of rows is the number of the image (training data) that the VGG16 extract from its feature
# we can take the number of rows by the first index in extracted feature shape by the VGG16
# and the number of columns is produced after reshaping the extracted features
# so it is determined by -1 (because we don't know the new produced dimensions after reshaping)
X_train_features = feature_extractor.reshape(feature_extractor.shape[0], -1)
```

Figure 13. Step2 in Feature extraction

- Save the output (extracted features) to use it as input to the SVM classifier

```
# This is our X input to SVM
X_for_SVM = X_train_features
```

Figure 14. Step3 in Feature extraction

3.3.2 Deepfake classification using (SVM)

- Build and Train SVM without Hyperparameter Tuning.

```
# Build the SVM with default parameters, and set the random state to keep the same results in each run
SVM_model = SVC(random_state = 42)

# Train the SVM  on training data (extracted features) with its labels
SVM_model.fit(X_for_SVM, y_train)

# Send test data through same feature extractor process
# to be used in SVM predction
X_test_feature = VGG_model.predict(X_test)
X_test_features = X_test_feature.reshape(X_test_feature.shape[0], -1)
```

Figure 15. Build and Train SVM without Hyperparameter Tuning

3.4 Model evaluation

Evaluate SVM without Hyperparameter Tuning

- Make prediction

```
###### Make predction on train dataset ######
y_train_pred = SVM_model.predict(X_train_features)

# Inverse le transform to get original label back (fake and real)
# Becouse we trained our SVM on integer labels (0 and 1)
y_train_predd = le.inverse_transform(y_train_pred)
y_trainn = le.inverse_transform(y_train)


###### Make predction on test dataset ######
y_test_pred = SVM_model.predict(X_test_features)


# Inverse le transform to get original label back (fake and real)
# Becouse we trained our SVM on integer labels (0 and 1)
y_test_predd = le.inverse_transform(y_test_pred)
y_testt = le.inverse_transform(y_test)
```

Figure 16. Make prediction on train and test dataset

- Accuracy, precision, recall and F1 score on train and test dataset

```
# calculate each evaluation matrics in training and testing prediction
# traning: to see the overfitting and underfitting
# testing: to see how well our model does

print('The training accuracy: ' + str(accuracy_score(y_trainn, y_train_predd)))
print('The testing accuracy: '+ str(accuracy_score(y_testt, y_test_predd)))

print('\n\n')

print('The traning precision: ' + str(precision_score(y_train, y_train_pred)))
print('The testing precision: ' + str(precision_score(y_test, y_test_pred)))

print('\n\n')

print('The traning recall: ' + str(recall_score(y_train, y_train_pred)))
print('The testing recall: ' + str(recall_score(y_test, y_test_pred)))

print('\n\n')

print('The traning F1: ' + str(f1_score(y_train, y_train_pred)))
print('The testing F1: ' + str(f1_score(y_test, y_test_pred)))
```

Figure 17. Accuracy, precision, recall and F1 score on train and test dataset

- Confusion Matrix

```
# Build the confusion matrix for label prediction in traning dataset
cm = confusion_matrix(y_trainn, y_train_predd)

# Build the display object that display the confusion matrix with coreesponding label name
cmd = ConfusionMatrixDisplay(cm, display_labels=['fake','real'])

# Plot the confusion matrix with title and axises names
cmd.plot()
cmd.ax_.set_title("Confusion matrix on traning")
cmd.ax_.set(xlabel='Predicted', ylabel='Actual')

# 0: fake, 1: real
# positive: fake, negative: real
```

Figure 18. Confusion matrix for train dataset

```
# Build the confusion matrix for label prediction in testing dataset
cm = confusion_matrix(y_testt, y_test_predd)

# Build the display object that display the confusion matrix with coreesponding label name
cmd = ConfusionMatrixDisplay(cm, display_labels=['fake','real'])

# Plot the confusion matrix with title and axises names
cmd.plot()
cmd.ax_.set_title("Confusion matrix on testing")
cmd.ax_.set(xlabel='Predicted', ylabel='Actual')

# 0: fake, 1: real
# positive: fake, negative: real
```

Figure 19. Confusion matrix for test dataset

3.5 Model improvement

Build and Train SVM with Hyperparameter Tuning using GridSearchCV

```
# Defining the common range of parameters values for the SVM model

# C: degree of correct classification. (higher value of C means a small margin = less misclassification)
# kernal: type of hyperplane used to separate the data ('linear'=linear hyperplane, 'rbf'=non linear hyper-plane)
# gamma: is a parameter for 'rbf' representing how exactly the model fits the training data
# high gamma value = perfectly fits the training data which leads to overfitting

param_grid = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001], 'kernel': ['rbf']},
 ]

# Define the scoring method using make_scorer()
# we have chosen the precision score
# becouse we force to reduce the FN prediction example
scorer = make_scorer(precision_score)

# SVM(): the model wihtout parameter value
# param_grid: list of parameters for the model with its range of values
# scoring: gird search will choose best model based on precsion score value
# n_jobs: to use the proceccors in parallel
# refit: after choose the best parammeter, retrain the model on all traning dataset(not just one cross-fold)
# verbose: controls the displayed message. higher value of verbose produce more message while run
grid = GridSearchCV(SVC(), param_grid, scoring=scorer, n_jobs=-1, refit = True, verbose = 3)
```

Figure 20. Build the GridSearchCV model

```
# fitting the model for grid search
grid.fit(X_for_SVM, y_train)

# print the best value of precesion score
print('Best Score: %s' % grid.best_score_)

# print best parameter after tuning
print("\nBest parameters set:")
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print("\nBest modelt:")
print(grid.best_estimator_)

# get the best estimtor model
best_model = grid.best_estimator_
```

Figure 21. Train the GridSearchCV model

Evaluate SVM with Hyperparameter Tuning

- It is the same code in evaluation section (3.4), but we applied the evaluation on the best gird search model.

3.6 Model prediction/inference

After we run code as in Figure 22 for eight times the result was as follows:

```
# Randomly we chose one image from testing data
n=np.random.randint(0, X_test.shape[0])
img = X_test[n]

# Disply chosen image on the screen
plt.imshow(img)

# preprocessing: resize
img = cv2.resize(img, (SIZE, SIZE))

# preprocessing: color space
img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

# preprocessing: normalization
img = img / 255.0

# Extract features from input_img
input_img_feature=VGG_model.predict(input_img)

# Resize input_img_feature to suti image size of SVM
input_img_features=input_img_feature.reshape(input_img_feature.shape[0], -1)

# predict on input_img afrer feature extraction
prediction_SVM = best_model.predict(input_img_features)[0]

# Reverse the label encoder to original name
prediction_SVM = le.inverse_transform([prediction_SVM])
y_testt = le.inverse_transform(y_test)

# Display prediction and actual classification on the secreen
print("The prediction for this image is: ", prediction_SVM)
print("The actual label for this image is: ", y_testt[n])
```

Figure 22. Test model with testing data

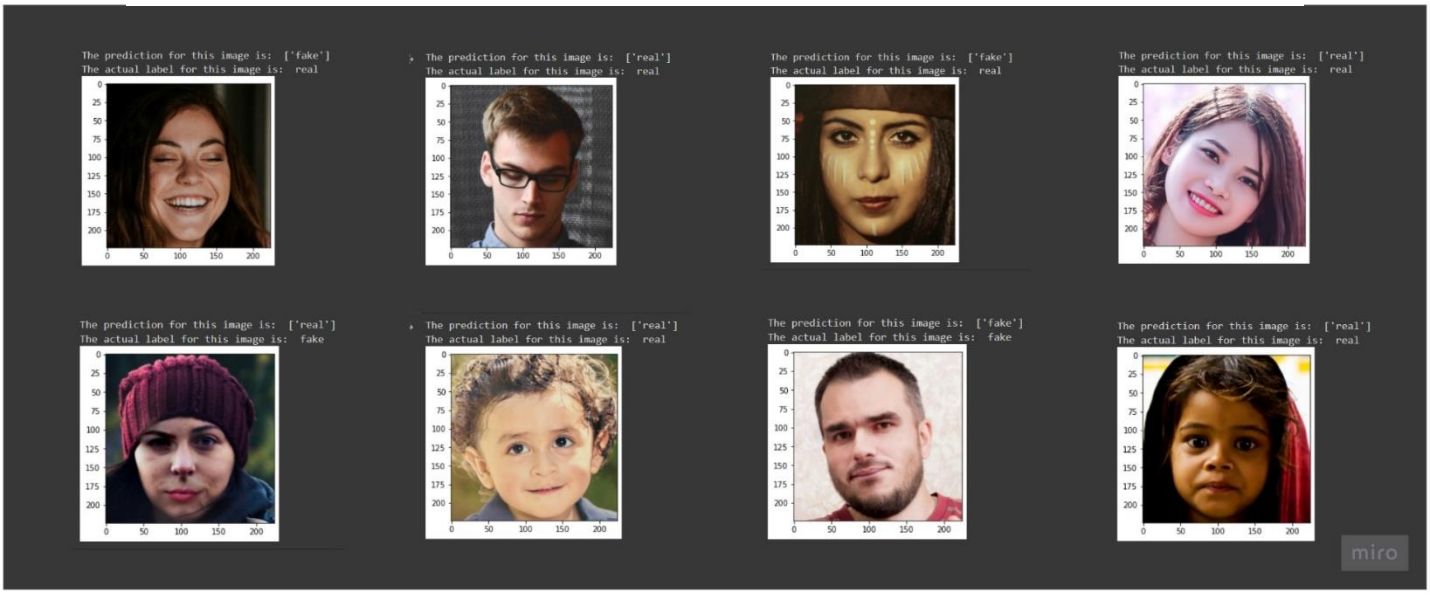


Figure 23. Prediction results

3.7 Save the model

We saved our model in a file to use for model deployment as in Figure 25. In our project, we used it again integrated with web page to create the interface, we illustrate the overall process in Figure.26.

```
# Save the model as a pickle in a file
joblib.dump(best_model, 'Deepfake detector.pkl')
```

Figure 24. Save model for later use

3.8 Model deployment

3.8.1 Overview about model deployment

A model deployment is a task that gives a chance for any end user to take benefit from the model. In our project, we trained our model then we saved it. After that we integrated the saved module with a web page to help the user to make sure of the images originality that he sees in social media the overall process shown in Figure.26.

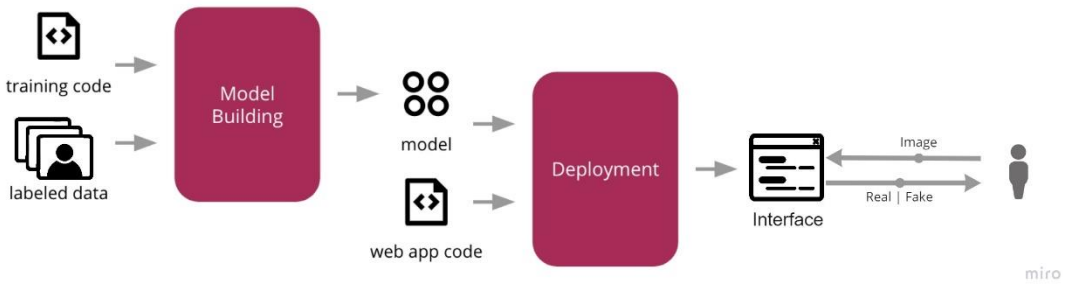


Figure 25. model deployment

3.8.2 Interface

The interface contains two buttons the first one is "Choose File" and the second one is "Submit" as shown in Figure 26. If the user wants to know if an image is fake or real, he must press "Choose File" button and select one image from his PC's and then he will press "Submit" button as shown in Figure 28. Our app will take the image and then return result as fake or real. The result depends on how an algorithm classified the image. The result will appear on the screen in the same web page and the image that we classified as shown in Figure 27.

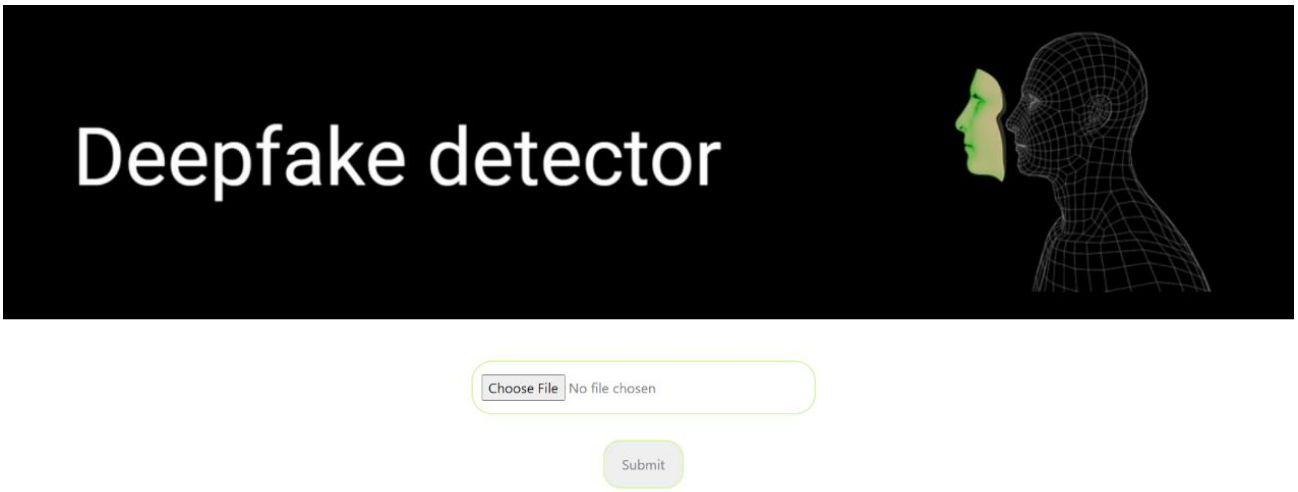


Figure 26. Interface Step 1 open Deepfake detector web page

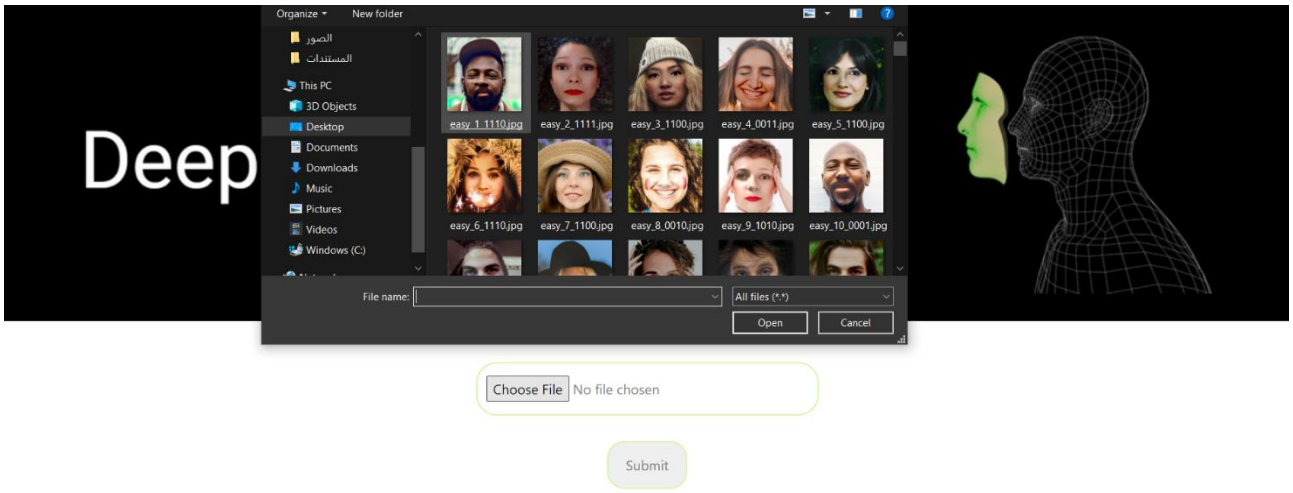


Figure 28. Interface Step 2 choose an image

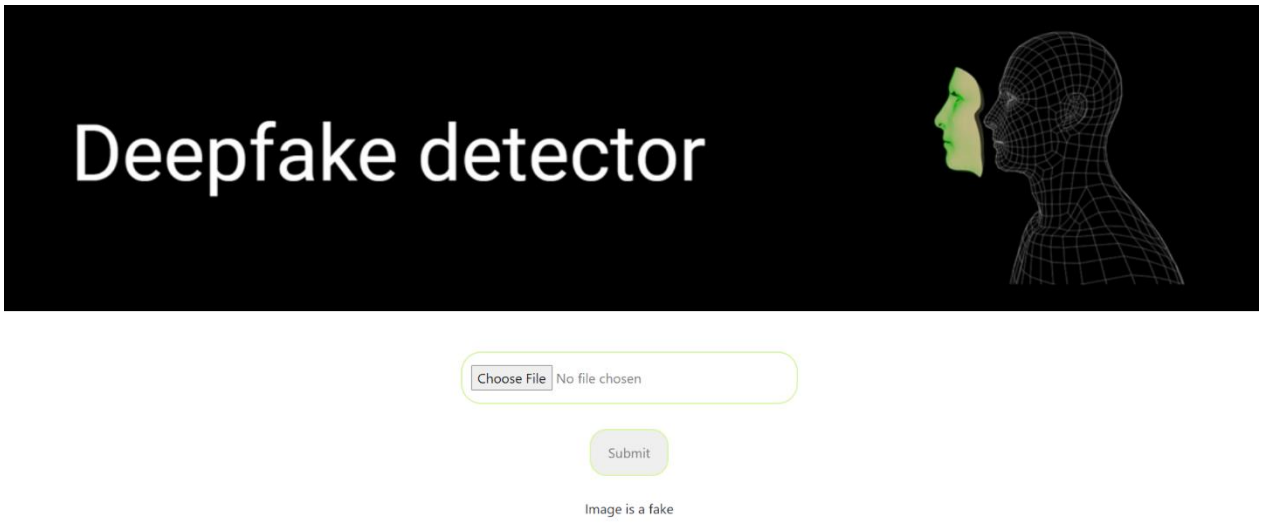


Figure 27. Interface step 3 display classification result on the screen

4. Discussion and Analyze the results

After exploring the results (both training and testing separately) with default vs hyper parameters in different metrics (i.e. precision, recall and F1) we have found that depending on the SVM with hyper parameter depending on **Precision** would be best fit. Which reduces the false negative. Trading off recall and focusing on the precision means that an algorithm returns more relevant results which is quality over quantity. In this section, we will illustrate result of our model three times:

- SVM with default parameters (without hyper parameter tunning).
- SVM with hyper parameter tunning by using GirdSearchCV to find the best model depend on **Precision** evaluation matrix.
- SVM with hyper parameter tunning by using GirdSearchCV to find the best model depend on **F1-score** evaluation matrix.

GridSearchCV runs through all the different parameters that is fed into the parameter grid and produces the best combination of parameters, based on a scoring metric of your choice (accuracy, f1, etc). [6]

4.1 SVM with default parameters (without hyper parameter tunning)

```
The training accuracy: 0.9893048128342246
The testing accuracy: 0.624


The traning precision: 0.9897959183673469
The testing precision: 0.6842105263157895


The traning recall: 0.9897959183673469
The testing recall: 0.5735294117647058


The traning F1: 0.9897959183673469
The testing F1: 0.6239999999999999
```

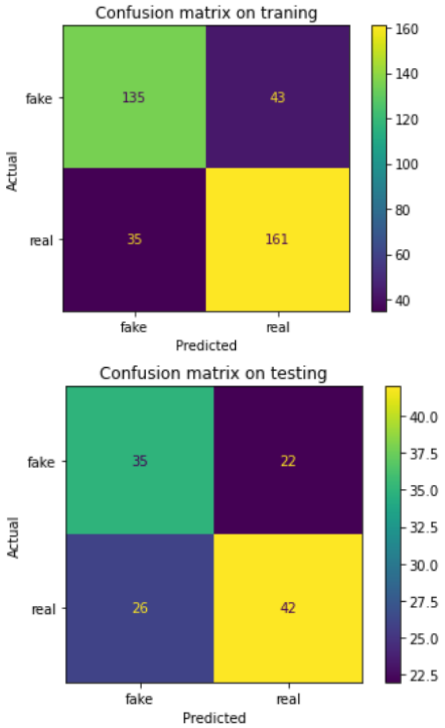


Figure 29. Result of SVM with default parameters

4.2 SVM with hyper parameter tunning by using GirdSearchCV to find the best model depend on *Precision* evaluation matrix.

```
The training accuracy: 0.7914438502673797
The testing accuracy: 0.616


The traning precision: 0.7892156862745098
The testing precision: 0.65625


The traning recall: 0.8214285714285714
The testing recall: 0.6176470588235294


The traning F1: 0.8049999999999998
The testing F1: 0.6363636363636364
```

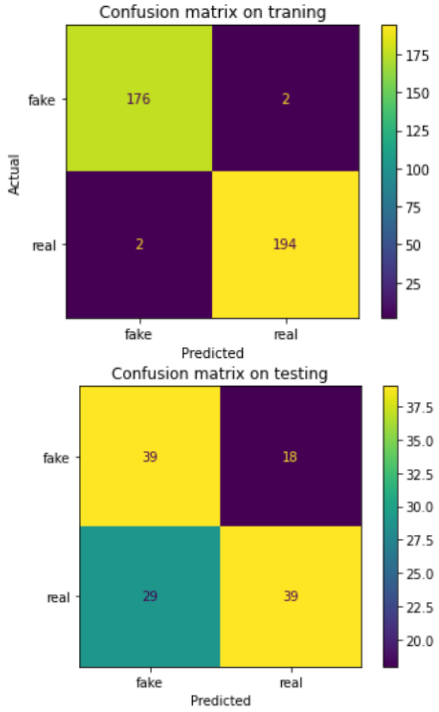


Figure 30. Result of SVM by using GridSearchCV based on precision evaluation metric

4.3 SVM with hyper parameter tunning by using GirdSearchCV to find the best model depend on *F1-score* evaluation matrix.

```
The training accuracy: 1.0
The testing accuracy: 0.544


The traning precision: 1.0
The testing precision: 0.544


The traning recall: 1.0
The testing recall: 1.0


The traning F1: 1.0
The testing F1: 0.7046632124352332
```

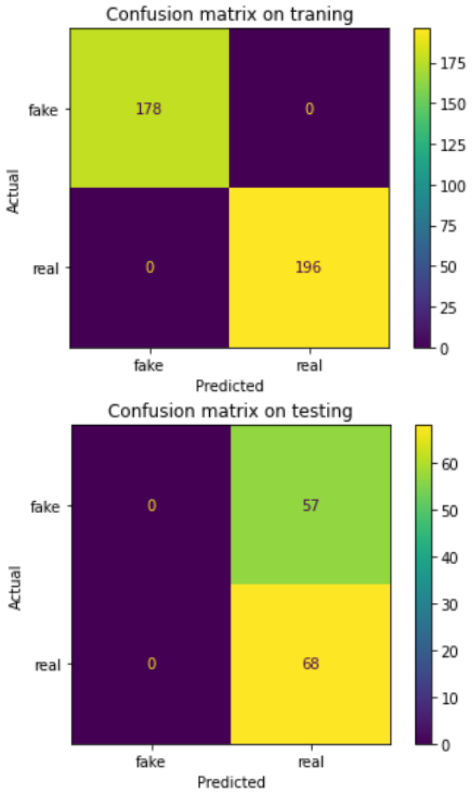


Figure 31. Result of SVM by using GridSearchCV based on F1-score evaluation metric

4.4 Discussion

In our project, we aim to reduce false negative (FN), which represents classifying fake image as a real image. Both **Precision** and **F1 score** reduce the FN. Therefore, we run the **GirdSeaerchCV** two times, the first time we set the score to **precision** (to choose the best SVM model based on the **precision score**) and the second time we set the score to **F1 score** (to choose the best SVM model based on the **F1 score**). We do that to compare between them and see which of them gives better results in test set and better result in train set as shown above in section 4.2-4.3.

- Run the **GirdSeaerchCV** on SVM based on **F1 score**, produce 100% value on training data for all evaluation matrices (overfit). For testing data, it produces a low percentage on most evaluation matrices.
- Whereas run the **GirdSeaerchCV** on SVM based on **Precision** score, produce 98% on training data for all evaluation matrices. For testing data, it produces a good percentage on most evaluation matrices.

Thus, we chose the **Precision** score to be used in **GirdSeaerchCV**.
The best parameter value and score the GirdSearchCV found for SVM are in Figure.30.

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Score: 0.6692360057571868

Best parameters set:
{'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}

Best modelt:
SVC(C=10, gamma=0.0001)
```

Figure 32. The best parameter values found by GridSearchCV

Important note, although we did some preprocessing steps and tried to improve the performance of the SVM, our model has overfitting problem, the current state-of-the-art deepfake detection method is still suffering from overfitting.

5. Tools and technologies

In our project, we use a lot of tools and technologies, we illustrate it in Figure 33.

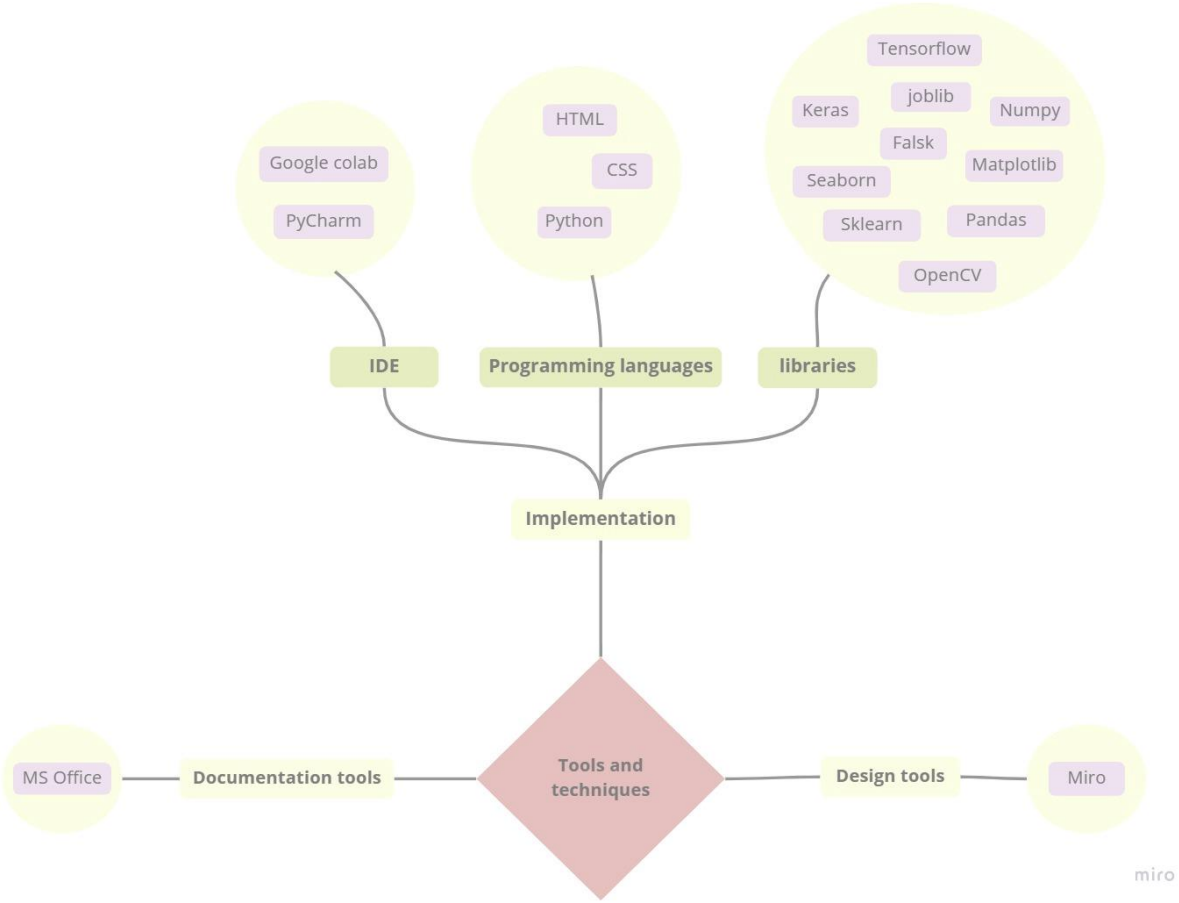


Figure 33. Tools and technologies used in project

6. Conclusion

In conclusion, since we satisfied the required plan, we have achieved what we were aiming to do. A high-precision deepfake detector using CNN(VGG16) as feature extractor and the supervised machine learning classifier SVM to classify with relatively high precision given our small dataset which was only 500 images. And we improved the SVM model by the GirdSeaerchCV technique to choose the best SVM model (best hyperparameter value). The GirdSeaerchCV evaluated based on many metrics then ended up choosing the precision score.

7. References

[1] Sreeni, Digital. *158b - Transfer Learning Using CNN (VGG16) as Feature Extractor and Random Forest Classifier*. YouTube, 22 Sept. 2020, https://www.youtube.com/watch?v=IuoEiemAuIY.

[2] Joy. “Machine Learning - YouTube.” *YouTube*, YouTube, 19 May 2021, https://www.youtube.com/playlist?list=PLBoXtbr1eVX-sjuH22-E2dUeAZKmY_Alv.

[3] Academy, Algorithm. “Flask - YouTube.” *YouTube*, YouTube, 28 Apr. 2020, https://www.youtube.com/playlist?list=PLfDx4cQoUNObFOYvkcNQG8arJX95TRE47.

[4] “Real and Fake Face Detection | Kaggle.” *Kaggle: Your Machine Learning and Data Science Community*, CIPLAB @ Yonsei University, 2019, https://www.kaggle.com/datasets/ciplab/real-and-fake-face-detection/code.

[5] “Transfer Learning in Keras with Computer Vision Models.” *Machine Learning Mastery*, https://www.facebook.com/MachineLearningMastery/, 14 May 2019, https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/.

[6] Okamura, Scott. “GridSearchCV for Beginners.” *Medium*, medium.com, 29 2020, https://towardsdatascience.com/gridsearchcv-for-beginners-db48a90114ee.