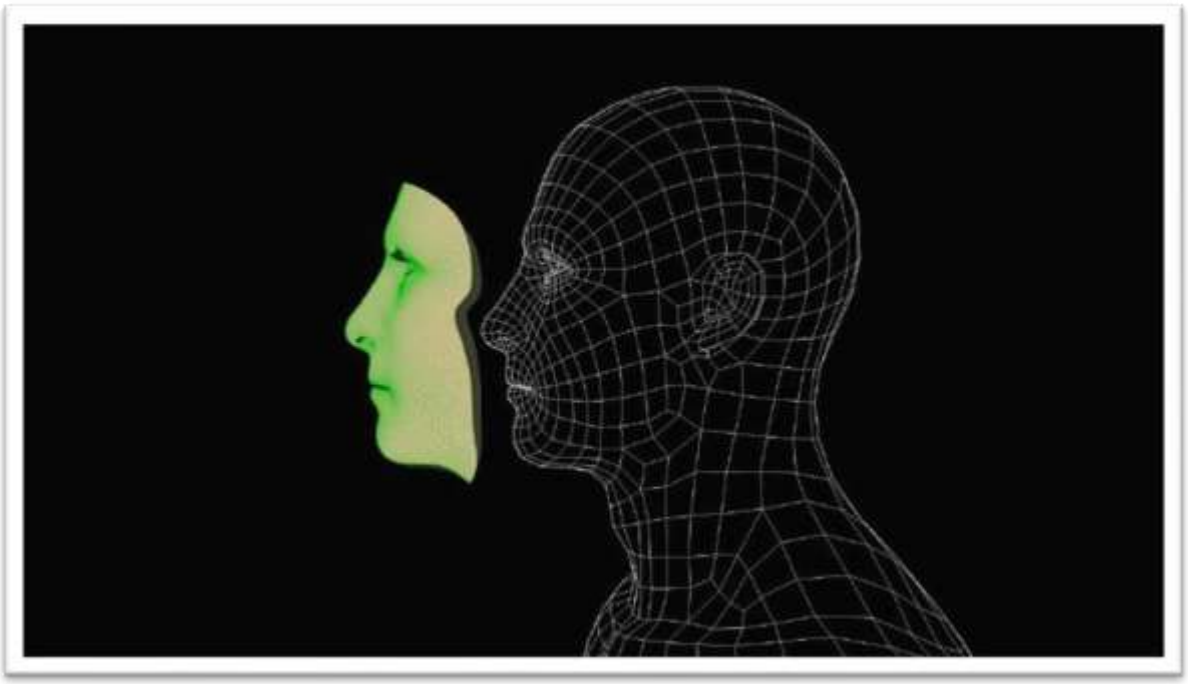
Al-Imam Muhammad Bin Saud Islamic University
College of Computer and Information Sciences,
Department of Computer Science
CS 467 Optimization and Metaheuristics
Course project
June 3th, 2023

Metaheuristic-Based Hyperparameter Tuning on Deepfake Video Detection using Jaya Algorithm



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

Content

1. Introduction 4
1.1 Problem Statement 4
1.2 Problem formulation 4
2. Proposed method 4
3.1. Celeb-DF (V2) Dataset 5
3.2. Dataset Pre-paring 5
3.2.1. Converting MP4 Videos to JPG Images (Frames) 5
3.2.2. Extract Faces from Frames 5
3.2.3. Connect Labels with Images 5
3.5. Dataset Pre-processing 5
3.6. Deepfake Detection Model 6
3.7. Hyperparameter Tuning using Optimization Method 6
3.7.1 Jaya Optimize Algorithm 6
4. Discussion and Analyze the results 8
4.1. Analyze the results 8
4.1.1 VGG-16 without hyperparameter tuning 8
4.1.2 VGG-16 with hyperparameter tuning using Jaya optimization algorithm. 9
4.2 Discussion 10
5. Tools and technologies 10
6. Conclusion 10
7. References 11

List of Figures

Figure 1 propsed model 4
Figure 2 Step of data preparation 5
Figure 3 Preprocessing step 6
Figure 4 Model without parameter tuning 6
Figure 5 Model with parameter tuning 6
Figure 6 Jaya algorithm flowchart 7
Figure 7 Result of model without parameter tuning 8
Figure 8 Result of model with parameter tuning 9
Figure 9 Tools and technologies used in project 10

List of Tables

Table 1 Jaua parameter initialization. 9
Table 2 Utilized hyperparameters range 9
Table 3 Optimized hyperparameters for the VGG-16 9
Table 4 Performance measures of models 10

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 enhance deepfake videos detection method based on Convolutional Neural Networks (CNNs) by using Jaya optimization algorithm.

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 a video 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 a video to a real or a fake depend on it is manipulated or not.
Experience (E)	A collection of real and fake faces videos.
Performance (P)	Classification evaluation metric, which are accuracy, recall, precision and F1.

2. Proposed method

The implementation for this project will be split into the following sections: dataset preparing; dataset pre-processing; deepfake detection model and hyperparameter tuning using optimization method. The overall model architecture that follows the step of implementation is illustrated in figure 1. We describing and explain our approach to tackling the CNN model by following the approach applied in this paper "Novel Convolutional Neural Networks based Jaya algorithm Approach for Accurate Deepfake Video Detection"[1]. We used their proposed method regarding the hyperparameter tuning of CNN using Jaya optimization algorithm.

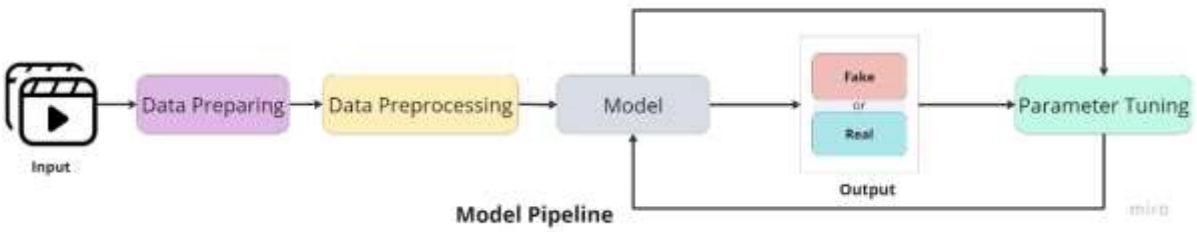


Figure 1 propsed model

3.1. Celeb-DF (V2) Dataset

We plan to detect fake videos by using Celeb-DF (v2) [2]. Celeb-DF (v2) is a large-scale challenging dataset for deepfake forensics. It includes 890 real MP4 videos and 5639 fake MP4 videos, total of 9 GB. The average length of all videos is approximate 13 seconds with the standard frame rate of 30 frame-per-second. The real videos are collected from YouTube with subjects of different ages, ethnic groups, and genders. The fake videos are generated by swapping faces. Dataset available on this link [3].

3.2. Dataset Pre-paring

Figure 2 shows the steps to perform data preparation. Each of these steps is explained in detailed in the following sections (3.2.1, 3.2.2. and 3.2.3.).

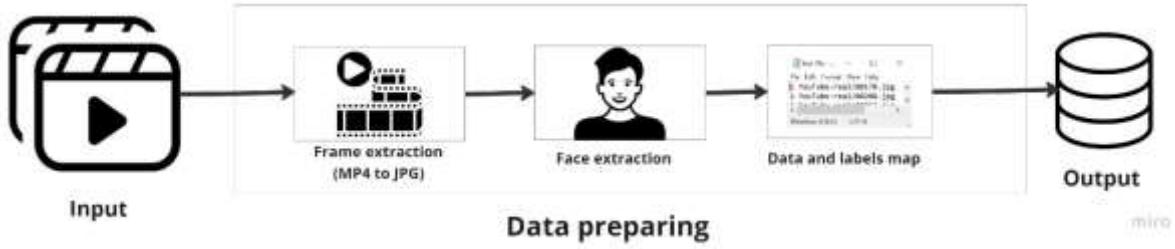


Figure 2 Step of data preparation

3.2.1. Converting MP4 Videos to JPG Images (Frames)

In order to make the dataset suitable to VGG-16 [4], we need to convert the video dataset into image dataset by extracting the video's frame. In this process, OpenCV's functions are used to load the video and to read the videos frame by frame. We choose to take only one frame per video because the person is existed in front of the image in most frames. Therefore, we don't need to save all frames of the video. It is important to note that we choose the frame on the second 5 because the first frames sometimes contain the background picture of the channel then the video is started with the person.

3.2.2. Extract Faces from Frames

The images extracted from the videos consist of a different noise, such as the background view of the person, the table that person sits at and so on. To prevent the VGG-16 from extracting features from these regions, all frames extracted must be cropped so the resulting image only contains the person's face. The MTCNN [5] module was used to detect such faces. The MTCNN shows its effectiveness to extract faces among many studies, including the deepfake detection studies. In addition, the model resizes the image after face extraction to the chosen size. The new image is then saved to the destination file path.

3.2.3. Connect Labels with Images

The dataset provides the label name to the test set named 'test labels.txt'. The format from these labels is preserved when generating the train set. In which the format contains two columns. The first one is the class name (1 for real and 0 for fake). And the second one is the file name of the image. By doing so, we can use these files when we load the dataset in an effective way.

3.5. Dataset Pre-processing

Firstly, the training and testing set are loaded with the filenames and its corresponding labels. The dataset is split into validation – train sets (15% - 85%) based on the common split value of previous deepfake detection studies in this dataset. The current splits (train, val and test) contain file names, not actual images. Therefore, a function is defined which can load the images from the file path and perform the necessary pre-processing. These necessary pre-processing are image resize, image normalization. Each split set is shuffled with its label. Hence, due to the memory limitations, we reduced each split to the half (50%) and used them to training and testing the model. As shown in

Figure 3

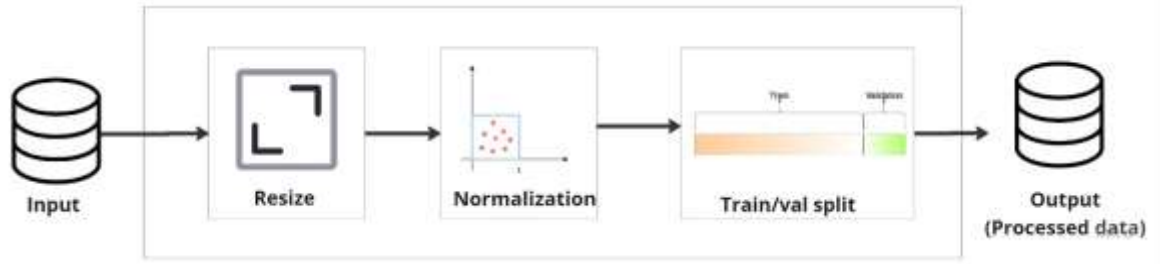


Figure 3 Preprocessing step

3.6. Deepfake Detection Model

Figure 4 shows the model architecture, we use VGG-16 as a pre-trained model that is trained on Image Net dataset. To make the VGG-16 suitable for our dataset, we use transfer learning to customize the VGG-16 for our task. Specifically, we use the VGG-16 as feature extractor in which all the pre-trained parameters are freeze and the last fully connected layers are removed and replaced with a new classifier to be trained from scratch depending on our task. This classifier consists of one dense layer (output layer) with the number of output node equal to one and with sigmoid activation function. Since our task output is either real or fake. We set the type of pooling to max depending on the previous studies. For the hole model (VGG-16 feature extractor and dense layer classifier), Adam optimizer with a learning rate of 0.0001 is used and binary cross entropy loss function. With batch size of 8.

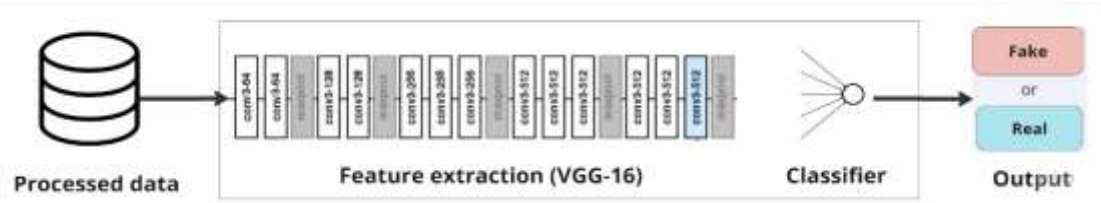


Figure 4 Model without parameter tuning

3.7. Hyperparameter Tuning using Optimization Method

In true machine learning fashion, we'll apply an optimization algorithm to perform exploration for hyperparameter tuning and select as much as possible the best value for the proposed model. We are inspired by this paper[1] that applies the Jaya optimization algorithm to hyperparameter tuning such as learning rate and number of epochs. The high-level approach of using Jaya with the basic model is illustrated in figure 5.

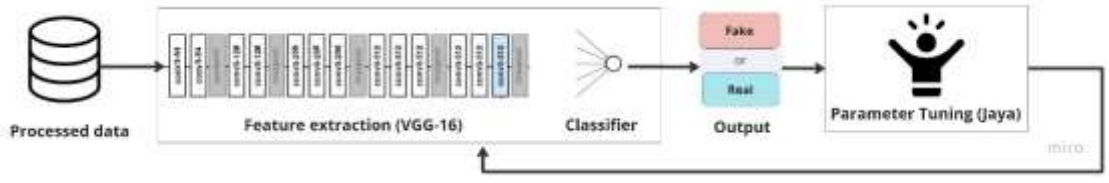


Figure 5 Model with parameter tuning

3.7.1 Jaya Optimize Algorithm

The Jaya algorithm was introduced by V. Rao in 2016 [7]. The algorithm strives to become victorious by reaching the best solution and hence it is named as Jaya (a Sanskrit word meaning victory). It is a population-based method that repeatedly modifies a population of individual solutions. but what distinguishes Jaya from the others is the fact that it does not contain any hyperparameters. The algorithm always tries to get closer to success (i.e. reaching the best solution) and tries to avoid failure (i.e. moving away from the worst solution). Depending on this mechanism, the searching agent updates basically its

position by keeping only the best position as well as precisely ignoring all worst positions. Hence, by updating all candidates of Jaya in each iteration, all solutions resulting from iteration are higher than the previous worst solution. Figure 6 shows the flowchart of the Jaya.

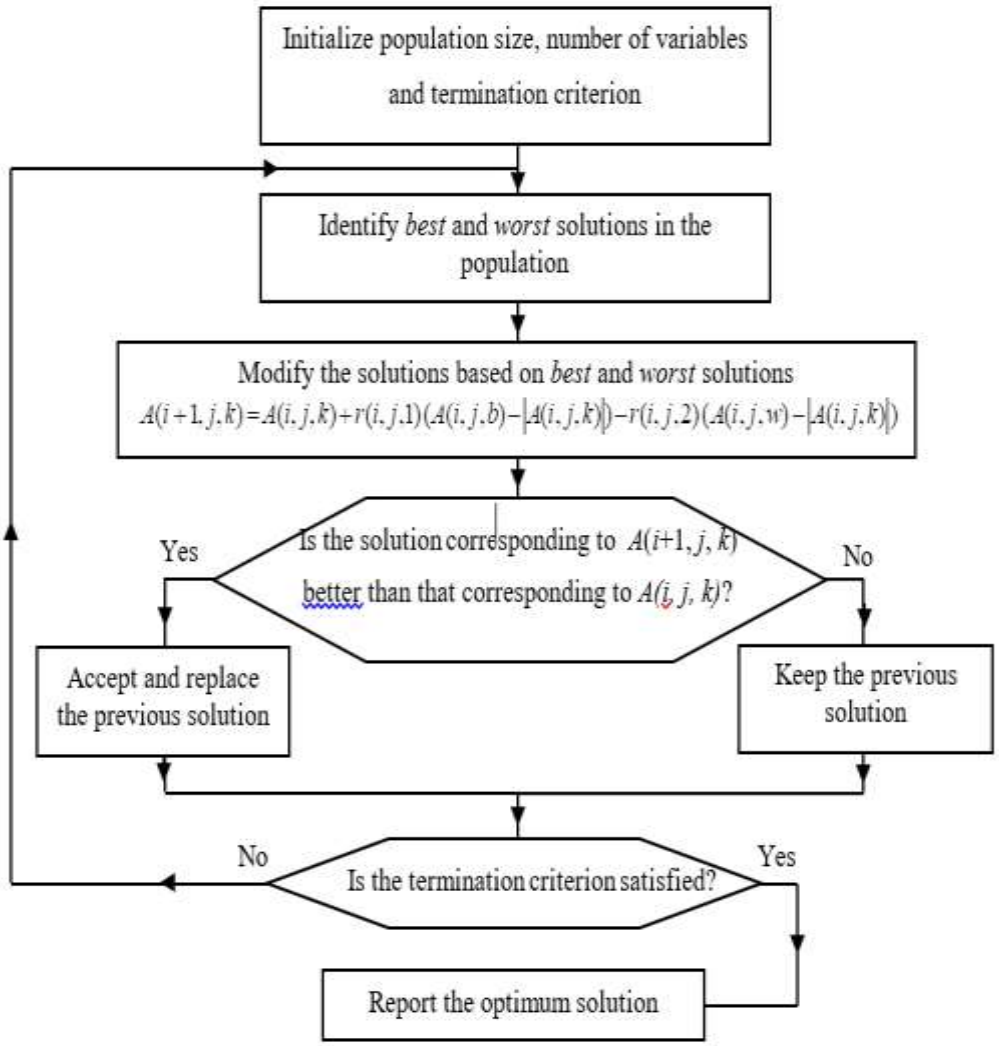


Figure 6 Jaya algorithm flowchart

In Jaya, every proposed solution or searching agent is known as a particle ($X_{j,k,i}$), every particle is seeking the most effective solution (J_{Best}) and keeps away from the worst solution (J_{worst}) of the objective or cost function (J), in the search area. This is achieved to optimize the objective function (J), supposing 'n' of candidates (i.e., k = 1, 2, ..., n) and 'm' of design variables or generators (i.e., j = 1, 2, ..., m). The positions of the particles are mathematically updated as (1) :

$$X'_{j,k,i} = X_{j,k,i} + r_{1,j,i}\left(X_{j_{Best},i} - |X_{j,k,i}|\right) - r_{2,j,i}\left(X_{j_{Worst},i} - |X_{j,k,i}|\right) \qquad (1)$$

where, $X_{j,k,i}$ $r$represents j^{th} variable for the k^{th} candidate during the i^{th} iteration. $r_{1,j,i}$ and $r_{2,j,i}$ are random numbers with values in the range [0, 1]. The best and worst candidate values are $X_{j_{Best},i}$ and $X_{j_{Worst},i}$ respectively. $X'_{j,k,i}$ are the new value of $X_{j,k,i}$. $X'_{j,k,i}$ value is kept if it returns the best fitness value compared to $X_{j,k,i}$. At the end of each iteration, the cost function values are maintained, and the next iteration considers these values as input for computation until the optimal solution is obtained. The pseudocode of Jaya, outlined in Algorithm 1.[8]

Algorithm 1 JA pseudo code.

Input:
n – Population size or number of candidates
m – Number of design variables or generators
Z – Maximum number of iterations
X – Population
Output:
J_{Best} or J_{min} – Best solution of the cost or fitness function (J)
J_{Worst} – Worst solution of the objective or fitness function (J)
X_{best} – Best candidate of the fitness function
X_{worst} – Worst candidate of the fitness function

Start
  for $k := 1$ to n **do** ▷ (i.e., population size)
    for $j := 1$ to m **do** ▷ (i.e., design variables)
      Initialize population ($X_{i,k}$)
    end for
  end for
  Evaluate $X_{best,1}$ and $X_{worst,1}$ ▷ (Based on the solution of the fitness function)
  Set $i = 1$ ▷ (Initialize iteration number)
  while maximum number of iterations (Z) is not met **do**
    for $k := 1$ to n **do** ▷ (i.e., population size)
      for $j := 1$ to m **do** ▷ (i.e., design variables)
        Set $r_{1,j,i}$ = a random number from [0, 1]
        Set $r_{2,j,i}$ = a random number from [0, 1]
        Update Eq. 1
      end for
      if solution $X'_{j,k,i}$ is better than $X_{j,k,i}$ **then** ▷ (Based on the solution of the fitness function)
        Set $X_{j,k,i+1} = X'_{j,k,i}$
      else
        Set $X_{j,k,i+1} = X_{j,k,i}$
      end if
    end for
    Set $i = i + 1$
    Update $X_{best,1}$ and $X_{worst,1}$
  end while ▷ maximum iterations or termination criterion satisfied
End

4. Discussion and Analyze the results

4.1. Analyze the Results

After exploring the results (both training and testing separately) in different metrics (i.e. accuracy, precision, recall and F1). In this section, we will illustrate result of our model two time:

- VGG-16 without hyperparameter tuning.
- VGG-16 with hyperparameter tuning using Jaya optimization algorithm.

4.1.1 VGG-16 without Hyperparameter Tuning

```
The training accuracy: 0.8786217697729053
The testing accuracy: 0.65625

The traning precision: 1.0
The testing precision: 1.0

The traning recall: 0.009584664536741214
The testing recall: 0.011235955056179775

The traning F1: 0.0189873417721519
The testing F1: 0.02222222222222222
```

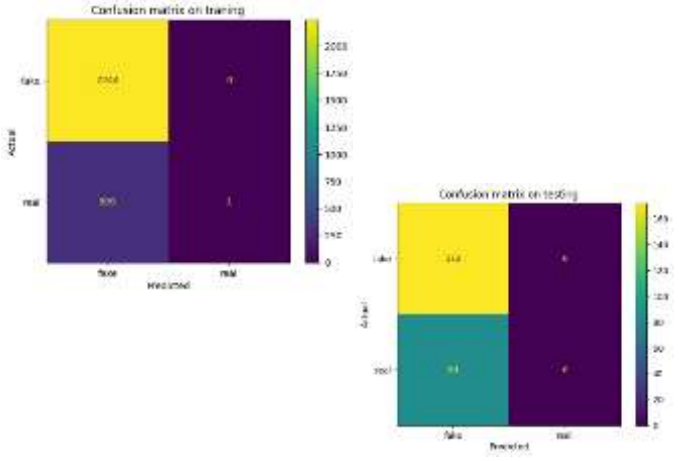


Figure 7 Result of model without parameter tuning

4.1.2 VGG-16 with hyperparameter tuning using Jaya optimization algorithm.

We have tested the Jaya optimization with different range of value for two hyperparameters which are the learning rate and number of epochs. The parameter setting for the Jaya an initially population of 5 individuals has been created and the number of generations considered is 5. as shown in Table1. The Table2 show parameters range to obtain optimal detector. In Table 3 lists the optimized hyperparameters obtained using Jaya optimization of the detector model.

Table 1 Jaya parameter initialization.

Parameter	value
Population size	5
Number of generations	5

Table 2 Utilized hyperparameters range

Hyperparameter	Range
Learning Rate	[0.01,0.0001], [0.01,0.09]
Number of epochs	1 to 50 with step 5

Table 3 Optimized hyperparameters for the detector

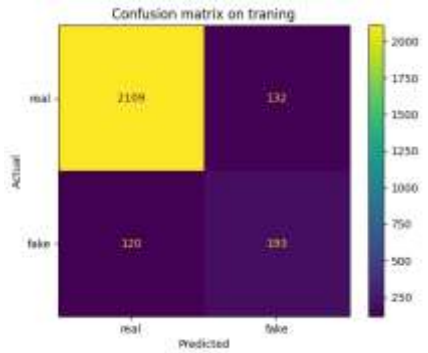
Hyperparameter	Range
Learning Rate	0.01
Number of epochs	15

```
The training accuracy: 0.9013312451057165
The testing accuracy: 0.7265625

The traning precision: 0.5938461538461538
The testing precision: 0.6938775510204082

The traning recall: 0.6166134185303515
The testing recall: 0.38202247191011235

The traning F1: 0.6050156739811912
The testing F1: 0.4927536231884058
```



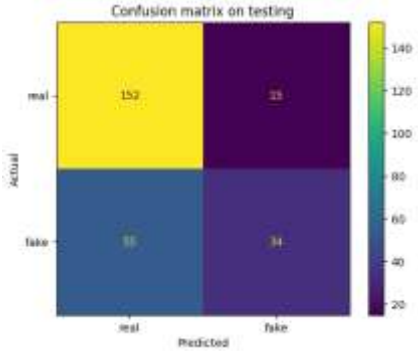


Figure 8 Result of model with parameter tuning

4.2 Discussion

Table 4 Performance measures of models

Model	Accuracy %	Recall%	Precision%	F1%
VGG-16	65.6	1.1	100	2.2
VGG-16+Jaya	72.6	38.2	69.3	49.2

Our project which applied the combination of the VGG16 architecture with the Jaya algorithm optimization, is highly effective in detecting deepfake videos. The model was evaluated on the Celeb-DF dataset. The proposed model achieved an accuracy of 72.6%, which is higher than the accuracy achieved by VGG16 without optimization which achieved an accuracy of 65.6%.the Table 4 show a comparison of performance measures to both model with optimization and without optimization.
Important note, although we did some preprocessing steps and tried to improve the performance of the VGG16, our model has an imbalance data problem and due to limitation of resources we don't set the parameter of Jaya optimization as in the inspired work [6].But we observe the effect of Jaya optimization in improve model accuracy by 10.6%.

5. Tools and technologies

In our project, we use a lot of tools and technologies, we illustrate it in Figure 9.

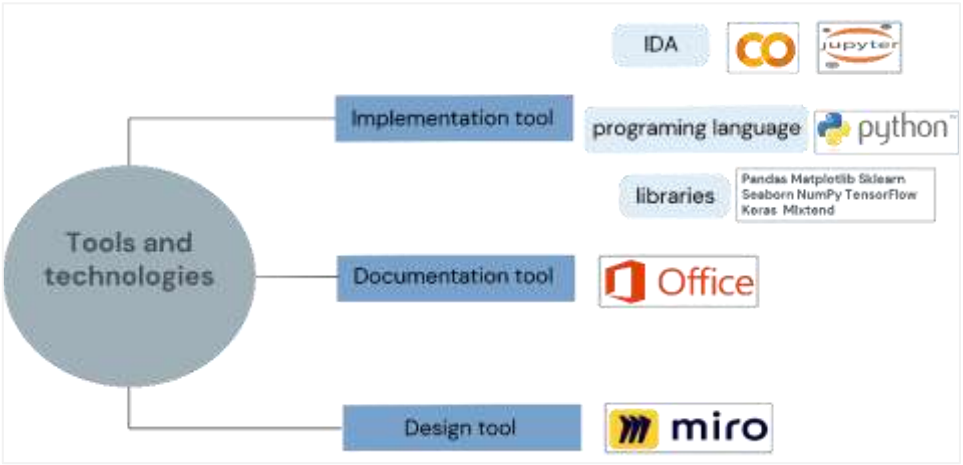


Figure 9 Tools and technologies used in project

6. Conclusion

In conclusion, since we satisfied the required plan, we have achieved what we were aiming to do. We applied an approach for detecting deepfake videos using the combination of the VGG16 architecture with the Jaya algorithm optimization. The approach was evaluated on the Celeb-DF dataset and achieved 72.6% performance. The results of the evaluation demonstrate that the proposed approach is highly effective when using Jaya optimization algorithm for Hyperparameter tuning. The Jaya algorithm optimization was used to optimize the hyperparameters of the VGG16 which are learning rate and number of epochs, which resulted in improved performance compared to VGG16 model without parameter tuning.

7. References

[1] Hussain, Zahraa Faiz, and Hind Raad Ibraheem. “Novel Convolutional Neural Networks Based Jaya Algorithm Approach for Accurate Deepfake Video Detection.” Mesopotamian Journal of Cyber Security, 24 Feb. 2023, pp. 35–39, https://doi.org/10.58496/mjcs/2023/007. Accessed 28 Feb. 2023.

[2] Li, Yuezun, et al. "Celeb-df: A large-scale challenging dataset for deepfake forensics." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020.

[3] Li, Yuezun. “Celeb-DF: A Large-Scale Challenging Dataset for DeepFake Forensics.” GitHub, 20 Apr. 2022, github.com/yuezunli/celeb-deepfakeforensics.

[4] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[5] Zhang, Kaipeng, et al. "Joint face detection and alignment using multitask cascaded convolutional networks." IEEE signal processing letters 23.10 (2016): 1499-1503.

[6] Z. F. Hussain and H. R. Ibraheem, “Novel Convolutional Neural Networks based Jaya algorithm Approach for Accurate Deepfake Video Detection,” Mesopotamian Journal of Cyber Security, pp. 35–39, Feb. 2023, doi: https://doi.org/10.58496/mjcs/2023/007.

[7] R. Venkata Rao, “Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems,” International Journal of Industrial Engineering Computations, pp. 19–34, 2016, doi: https://doi.org/10.5267/j.ijiec.2015.8.004.

[8] E. H. Houssein, A. G. Gad, and Y. M. Wazery, “Jaya Algorithm and Applications: A Comprehensive Review,” Lecture Notes in Electrical Engineering, pp. 3–24, Nov. 2020, doi: https://doi.org/10.1007/978-3-030-56689-0_2.

[9] D. Rai, “Jaya Optimization Algorithm,” Medium, Jan. 08, 2019. https://dhiraj-p-rai.medium.com/jaya-optimization-algorithm-16da8708691b (accessed May 14, 2023).