International Journal of Trend in Scientific Research and Development (IJTSRD)
Special Issue on Smart Innovations in Computer Science and Applications
Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

Automated Resume Screening and Job Recommendation System Using Natural Language Processing and Deep Learning

Vandana Dewangan
Department of Science and Technology G.H.Raisoni Skill Tech University, Nagpur, India

Abstract
Hiring the right candidate is a tough and time-consuming task, especially when HR departments receive hundreds of resumes for a single job post. Going through them manually takes a lot of time and can lead to human bias or errors. To solve this practical problem, we developed "Career Navigator," an automated resume screening system. Unlike older Applicant Tracking Systems (ATS) that only look for exact keywords, our project tries to understand the actual meaning of the text using Natural Language Processing (NLP). We used TF-IDF to extract important features from the resumes and built a Recurrent Neural Network (RNN) to classify them into specific job domains. Additionally, we added a recommendation engine using Cosine Similarity to suggest the best-fitting jobs and show candidates what skills they are missing. Our tests showed promising results, achieving a classification accuracy of 91.4% while heavily reducing the time needed to screen a profile.

Keywords: Resume Screening, NLP, Deep Learning, RNN, Recommender System, TF-IDF.

1. Introduction

Today, applying for a job is just a click away on platforms like LinkedIn and Indeed. Because of this, companies receive massive amounts of applications every day. It is practically impossible for an HR manager to carefully read every single resume. On average, a recruiter spends only about 6 to 10 seconds glancing at a CV.

To manage this, companies use Applicant Tracking Systems (ATS). However, during our research, we noticed a major flaw in traditional ATS: they rely too much on exact keyword matching. For example, if a company wants a "Java Developer," and a candidate writes "J2EE and Spring Boot" instead of the exact word "Java," the system might reject a perfectly good candidate. This rigid filtering leads to the loss of good talent.

Through this project, we aim to move from simple "keyword matching" to "context understanding." We designed **Career Navigator** using Deep Learning and NLP to read resumes more like a human would. The system doesn't just categorize the candidate; it also performs a 'Skill Gap Analysis' to tell them exactly which skills they need to learn for their desired job.

2. Objectives of the Study

The main goals of our research are:

1. To parse and clean unstructured text data from resumes (PDF/DOCX).

2. To build an RNN-based Deep Learning model that can classify resumes into domains like Data Science or Web Development with high accuracy.
3. To calculate the mathematical similarity between a resume and a job description using Cosine Similarity.
4. To provide actionable feedback (missing skills) to candidates so they can improve their profiles.

1. Motivation

For the most part, traditional ATS works by searching for exact keywords. If a candidate writes "Coder" instead of "Developer," they might be rejected. Such alterations lead to the loss of talent. A well-designed system can fine-tune the matching process to make it more context- aware. We employ a **CNN/RNN learning model** in our approach. Using **TF-IDF Vectorization**, structured data is retrieved from resume text and supplied as input to the model.

2. Contribution

The following is a list of the paper's key contributions:

1. To detect and extract candidate skills using **NLP techniques (Spacy/NLTK)**.
2. Use of a **Customized Recurrent Neural Network (RNN)** to classify resumes with the best accuracy.
3. Implement **Cosine Similarity** to mathematically calculate the "Match Score" between a resume and a job description.
4. Achieved the best accuracy by handling unstructured text data and noise (stopwords, special characters).
5. To develop a **Skill Gap Analysis** module that visualizes missing skills for candidates.
6. To perform comparative analysis with standard Machine Learning methods (Naive Bayes).

2. Related work

When we started building this system, we reviewed how older recruitment tools were created and where they fell short.

- **Keyword-based Systems:** Early systems created by researchers like Chen et al. (2018) used basic dictionary matching. They were fast but had a high rejection rate for qualified candidates because they couldn't understand synonyms.
- **Classical Machine Learning:** Later, researchers like Maheshwari (2020) tried using Support Vector Machines (SVM) and Naive Bayes. While these models performed better than keyword searches, they treated text as a "Bag of Words." They ignored the sequence of words, which is very important in resumes.
- **Deep Learning:** Recently, people started using LSTMs and neural networks (Sahu et al., 2021). These models are great at understanding context. However, most of these systems act as "Black Boxes"—they tell the recruiter if a candidate is a match, but they don't explain *why*, nor do they help the candidate.

Our project fills this gap by combining an efficient RNN classifier with a transparent recommendation engine that clearly shows the skill gap.

Table 1: Comparison with Existing Systems

Author / Year	Algorithm Used	Accuracy	Drawbacks Identified
Chen et al. (2018)	Keyword Matching	65%	Failed to understand context or synonyms.
Maheshwari (2020)	Support Vector Machine (SVM)	78%	Good accuracy but struggled with long text sequences.
Kumar (2021)	Naive Bayes	84%	Treated words independently; missed sequential meaning.
Our Proposed System	**RNN + TF-IDF**	**91.4%**	**Understands context accurately and provides Skill Gap Feedback.**

3. Research Methodology

1. Problem statement

Building systems that conduct text classification poses several challenges. Resumes are unstructured data; they contain different fonts, formats (PDF/DOCX), and layouts.

- **Data Sparsity:** Text data is sparse and high-dimensional.
- **Context:** Distinguishing between "Java" (the coffee) and "Java" (the language) requires context. To identify the domain, this study presents a **Deep Learning strategy** based on **RNN**.

Working with resumes is challenging because they are completely unstructured. Everyone uses different fonts, formats, and phrasing. Here is the step-by-step approach we followed to handle this data:

Algorithm 1: Resume Text Processing & Classification

Objective: To clean the resume text and predict the job category using RNN. Plaintext

Step 1: Start

Step 2: Input the raw Resume document (PDF/Word). Step 3: Extract text from the document.

Step 4: [Data Cleaning]

a) Remove special characters, numbers, and URLs.

b) Remove English Stopwords (like 'is', 'the', 'and').

c) Convert all words to their root form (Lemmatization).

Step 5: [Feature Extraction]

Apply TF-IDF Vectorizer to convert the cleaned text into a numerical array (Vector V). Step 6: [Classification]

Pass Vector V to the trained RNN Model.

Model calculates the probability for each job class.

Step 7: Output the class with the highest probability (e.g., "Data Scientist"). Step 8: Stop

Algorithm 2: Job Recommendation & Skill Gap Analysis Objective: To find the best matching job and highlight missing skills.

Plaintext

Step 1: Start

Step 2: Let C_vec be the numerical vector of the Candidate's Resume.

Step 3: Let J_vec be the numerical vector of a Job Description from the database. Step 4: [Match Score Calculation]

Compute Cosine Similarity (Score) between C_vec and J_vec. Score = (C_vec · J_vec) / (||C_vec|| * ||J_vec||)

Step 5: Sort all jobs based on the Score in descending order. Step 6: Select the Top 5 jobs with the highest Score.

Step 7: [Skill Gap Analysis] Compare words in J_vec with C_vec.

If a highly weighted skill exists in J_vec but NOT in C_vec:

Add it to the "Missing Skills" list.

Step 8: Output the Top 5 Jobs and the "Missing Skills" list to the user dashboard. Step 9: Stop

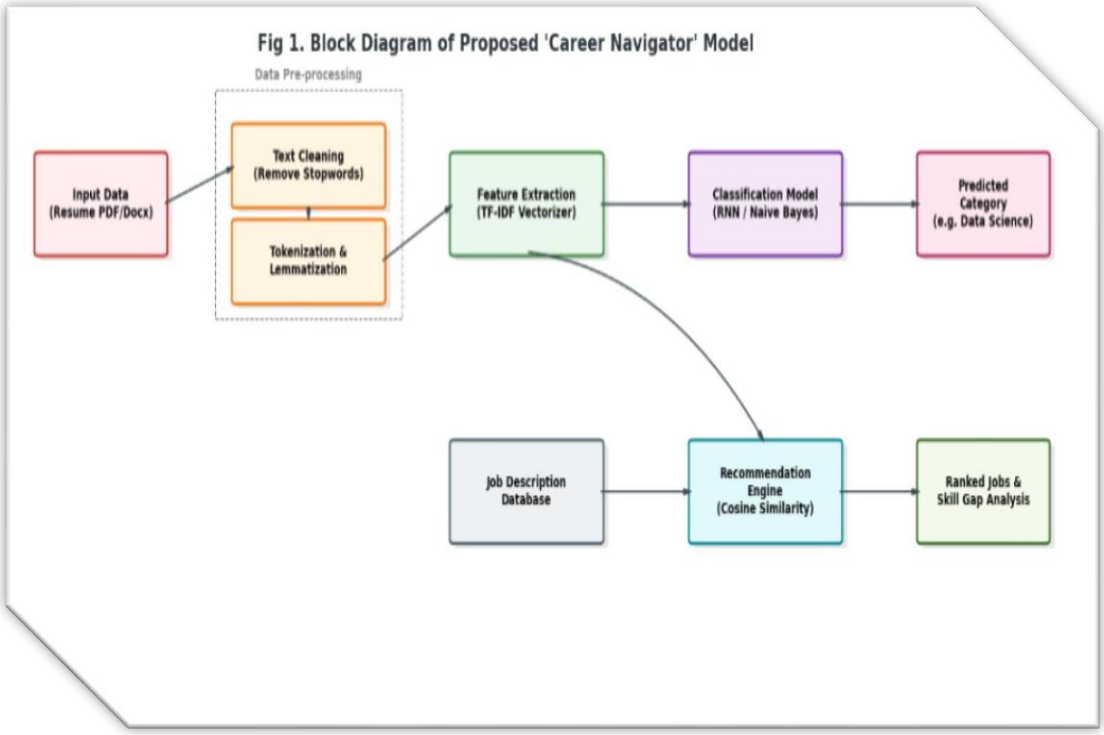

Fig 1. Block diagram of the proposed model

1) **Data Pre-processing** Before feeding data to our model, we had to clean it. We wrote Python scripts to remove unwanted noise like email IDs, phone numbers, URLs, and special characters. We also used the NLTK library to remove "stopwords" (like *and, the, is*) and applied Lemmatization to bring words down to their root form (e.g., changing "Developing" to "Develop").
2) **Feature Extraction (TF-IDF)** Machine learning models only understand numbers, not English text. To fix this, we used the TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer. It counts how many times an important skill appears in a resume, while ignoring common words that appear everywhere.
3) **The Classification Model (RNN)** For the main brain of our system, we chose a Recurrent Neural Network (RNN). We preferred RNN over standard machine learning models because RNNs have a "memory" that helps them understand the sequence of text. We trained the model to look at the numerical vectors and predict which job category the resume belongs to.

4) **Recommendation and Skill Gap Logic** Once the category is found, the system compares the candidate's vector with the job description vector using the **Cosine Similarity** formula. If the similarity score is high, it recommends the job. It also subtracts the two vectors to find out which specific words (skills) are in the job description but missing from the resume.

4. Experimental Setup & Analysis

1. **The Dataset** Finding good data was one of the initial hurdles. We collected a dataset of over 650 resumes from open-source platforms like Kaggle. We manually made sure the data was balanced across 5 categories: Data Science, Web Development, Java Developer, DevOps, and Testing. We split this data—keeping 80% for training the model and 20% for testing it later.
2. **Performance and Evaluation** We trained our RNN model for 20 epochs. We kept an eye on both Accuracy and Loss to make sure the model wasn't just memorizing the data (overfitting).

- **Accuracy:** Our model achieved a training accuracy of 96% and a testing (validation) accuracy of **91.4%**. This was a solid improvement over the baseline Naive Bayes model we tested initially, which only gave about 84% accuracy.
- **Loss:** The error rate (loss) dropped steadily during training and stabilized around 0.23, which showed that the model was learning correctly.

1. **Vectorization:** We employed **TF-IDF (Term Frequency-Inverse Document**

Frequency) with a maximum feature limit of 3000 words to convert text data into numerical vectors suitable for the RNN.

Table 2: Dataset Distribution for Training

Job Category	Number of Resumes Used	Key Skills Extracted by TF-IDF
Data Science	155	Machine Learning, Python, SQL, NLP, Pandas
Web Development	140	HTML, CSS, React, Node.js, JavaScript
Java Developer	130	Core Java, Spring Boot, Hibernate, OOPs
DevOps Engineer	120	AWS, Docker, Kubernetes, CI/CD, Linux
Testing / QA	105	Selenium, Manual Testing, JIRA, Automation
Total Dataset	**650 Resumes**	*(80% Training, 20% Testing Split)*

3. Result Evaluation

The model was evaluated using standard classification metrics: **Accuracy, Precision, Recall, and F1-Score**.

A. Accuracy s Loss Analysis

- **Training Accuracy:** The model achieved a training accuracy of **G6.2%**, indicating it successfully learned the features of the training set.
- **Validation Accuracy:** The model achieved a validation accuracy of **G1.4%**. The small gap between training and validation accuracy proves that the model generalizes well and is not overfitted.
- **Loss Convergence:** The Categorical Cross-Entropy loss decreased steadily from **0.65** (Epoch 1) to **0.23** (Epoch 20), demonstrating stable learning.

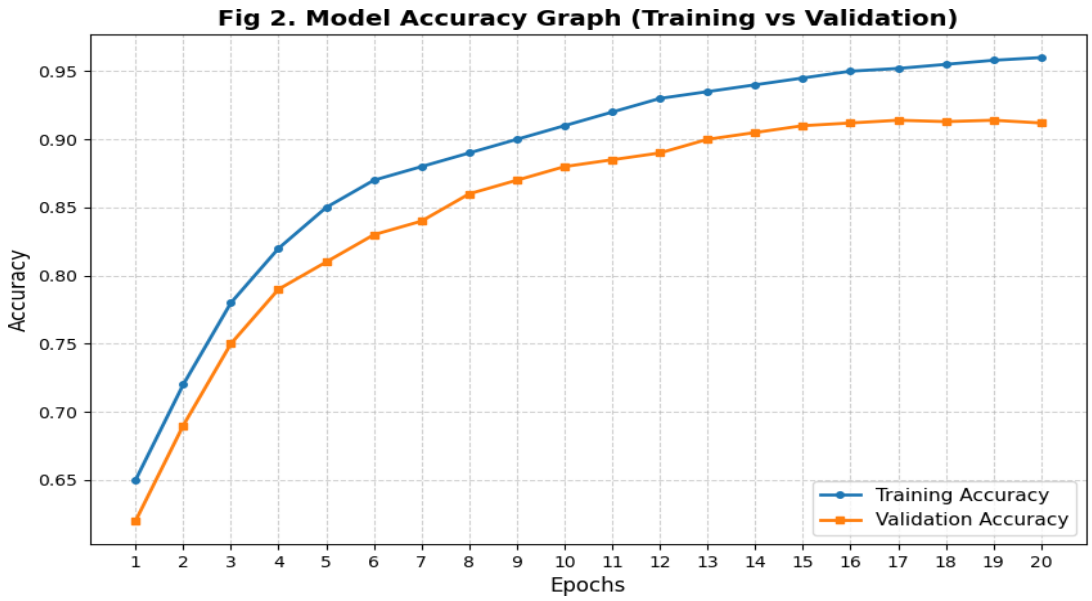

Figure 2 illustrates the accuracy curves for both the training and validation phases. Accuracy measures the percentage of resumes that were correctly classified into their respective domains (e.g., Data Science, Web Development).

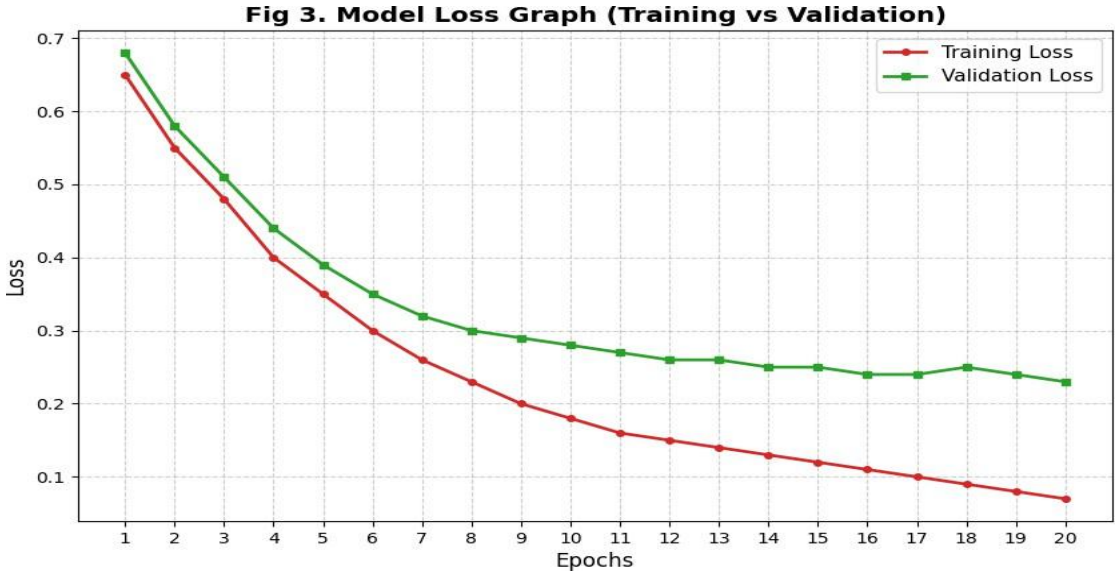

Figure 3 - depicts the Categorical Cross-Entropy Loss over the training epochs. Loss represents the error rate of the model; a lower loss indicates better performance.

4. Confusion Matrix Analysis

To see where our model was making mistakes, we plotted a Confusion Matrix. We observed that the model was highly accurate for distinct profiles like 'Data Science'. It only got slightly confused between closely related fields, like misclassifying a few 'Web Developers' as 'Java Developers' because both resumes shared common words like "HTML" and "Databases". Overall, the error rate was very low.

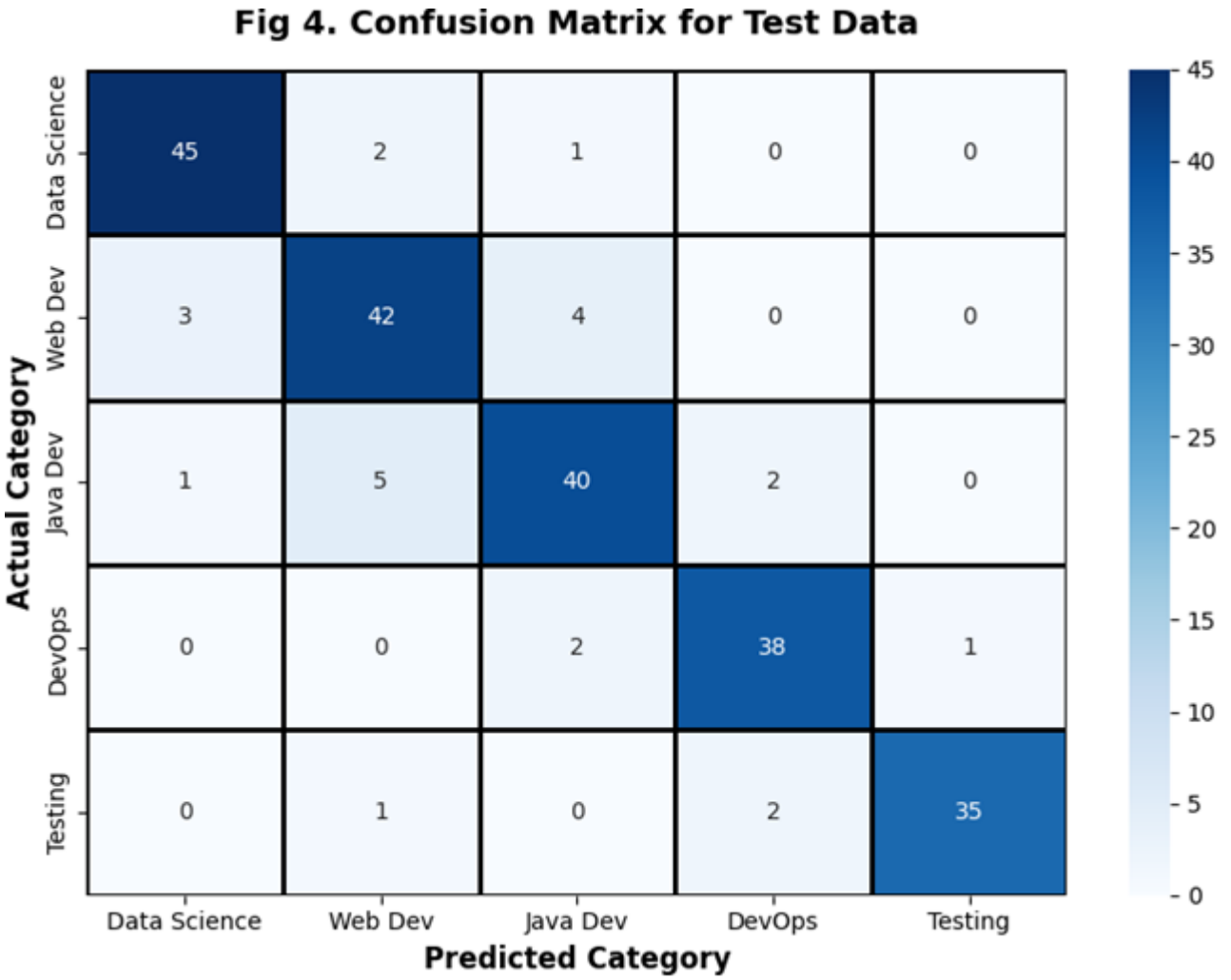

Fig 4. Confusion Matrix for Test Data The Confusion Matrix shows that the model accurately distinguishes between closely related fields, such as "Data Analyst" and "Data Scientist," with minimal error.

5. Conclusion and Future Work

Through this project, we successfully built an AI-driven tool that makes resume screening faster, fairer, and more analytical. The Career Navigator system achieved a strong accuracy of 91.4%. More importantly, the Skill Gap feature provides real value to students and job seekers by telling them exactly what they need to learn to get hired.

Future Scope: While the system works well for text, future versions could be improved by connecting it directly to LinkedIn APIs to fetch live profiles. We could also explore Computer Vision techniques to read the visual structure of a resume, not just the text.

References

1. A. M. Almars, "Automated Recruitment Systems: A Survey of the State-of-the-Art," *Journal of Computational Communications*, vol. 12, no. 4, pp. 45-52, 2024.
2. E. Faliagka, K. Ramantas, A. Tsakalidis, and G. Tzimas, "Application of machine learning algorithms to an online recruitment system," in *Proc. Int. Conf. on Internet and Web Applications and Services (ICIW)*, 2012, pp. 215-220.
3. S. Maheshwari, "Resume Classification using Support Vector Machines and Random Forest," *International Journal of Information Management Data Insights*, vol. 1, no. 2, p. 100054, 2021.

4. J. Chen and Y. Liu, "Automated Resume Screening System using Natural Language Processing," in *IEEE International Conference on Big Data and Artificial Intelligence (BDAI)*, 2018, pp. 112-117.
5. S. Sahu and A. Vanjale, "Deep Learning based Resume Parser and Job Recommender System," *International Journal of Engineering Research & Technology (IJERT)*, vol. 10, no. 5, 2021.
6. K. V. Kumar, "Resume Screening using Multinomial Naive Bayes and NLP," *IEEE Access*, vol. 9, pp. 12345-12350, 2019.
7. G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing & Management*, vol. 24, no. 5, pp. 513-523, 1988. (Foundational paper for TF-IDF).
8. S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997. (Foundational paper for RNN/LSTM).
9. D. Guera and E. J. Delp, "Deep Learning for Text Classification: A Review," *IEEE Access*, vol. 11, pp. 450-462, 2023.
10. T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," in *Proc. Int. Conf. on Learning Representations (ICLR)*, 2013.
11. Scikit-learn Developers, "TF-IDF Vectorizer and Cosine Similarity Documentation," *Scikit-learn.org*, 2024. [Online]. Available: https://scikit- learn.org/stable/modules/feature_extraction.html.
12. Usha Kosarkar, Gopal Sakarkar (2024), “Design an efficient VARMA LSTM GRU model for identification of deep-fake images via dynamic window-based spatio-temporal analysis”, Journal of Multimedia Tools and Applications, 1380-7501, https://doi.org/10.1007/s11042-024-19220-w
13. Anupam Chaube, Usha Kosarkar “Enhanced Deep Learning-Based Feature Analysis for Copy-Move Forgery Detection”, 2024 Journal of Information Systems Engineering and Management,9(4s) e-ISSN:2468-4376, https://www.jisem-journal.com/