Detection and Mitigation of Deepfake Attacks in Cybersecurity: Leveraging Computer Vision and Deep Learning

Bachelor's Thesis
Degree Programme in Computer Applications
Spring 2025
Poorya Zare Janakbari

HAMK
Häme University
of Applied Sciences

Abstract

DP	Degree Programme in Computer Applications
Author	Poorya Zare Janakbari Year 2025
Subject	Detection and Mitigation of Deepfake Attacks in Cybersecurity: Leveraging Computer Vision and Deep Learning
Supervisors	Mazhar Mohsin

In recent years, the production of fake content has grown exponentially, challenging cybersecurity, digital integrity, and public trust. The thesis presented here attempts to implement and evaluate a deepfake detection system that uses computer vision and deep learning techniques to classify real or manipulated facial images. In this research, three architectures are investigated (EfficientNetB4, DenseNet201, and EfficientNetV2L), each of which is a feature extractor, combined with a custom-made Dense Neural Network (DNN) that performs binary classification.

The proposed system was first trained and then tested on a balanced dataset of 140,000 images in two categories, real and fake. The system used various preprocessing and augmentation techniques to improve generalization. The system was trained on both Google Colab Pro and Pohti supercomputers to verify and ensure that the models are capable of running under different hardware conditions.

Accuracy, precision, recall, and F1-score were implemented to evaluate the system performance, which was supported by visualizations such as training curves and confusion matrices. The findings show that EfficientNetV2L + DNN has the best performance in image recognition, followed by EfficientNetB4 + DNN with remarkable accuracy. DenseNet201 + DNN faced an overfitting problem and poor performance on the test set.

The results of this study suggest that lightweight pretrained convolutional neural networks combined with dense classifiers can best detect fake content. Finally, this research proposes a comparative perspective on deep learning architectures for manipulated media analysis that can be simultaneously applied to identity verification and digital forensics.

Keywords	Deepfake, Cybersecurity, Computer Vision, Convolutional Neural Networks, Deep Learning, Feature Extraction.
Pages	40 pages and appendices 4 pages

Table of Contents

1 Introduction 1
2 Theoretical Foundations and Conceptual Framework 2
2.1 Deepfake Detection Models 3
2.2 Convolutional Neural Networks and Feature Extraction 3
2.3 Data Mining and Deep Learning in Cybersecurity 4
2.4 Deepfake Generation Techniques 5
2.4.1 Generative Adversarial Networks (GANs) 6
2.4.2 Autoencoders 6
2.4.3 Face Swapping 7
3 Deep Learning and Computer Vision in Deepfake Detection 8
3.1 Convolutional Neural Networks (CNNs) in Deepfake Detection 8
3.1.1 Pretrained CNN models to identify deepfake 8
3.1.2 Comparative performance of CNNs 9
3.2 Optimization Models and Transfer Learning 9
3.3 The Role of Computer Vision and Feature Extraction 10
3.4 Overall Framework of a Deepfake Detection System 11
3.5 Common Challenges in Deepfake Detection 12
4 Methods 13
4.1 Research Design 13
4.2 Dataset Description 14
4.3 Tools and Libraries Used 15
4.4 Model Architecture and Configuration 16
4.5 Evaluation Metrics 17
5 Practical Implementation 17
5.1 System Setup and Environment 18
5.2 Feature Extraction Using Pretrained CNNs (EfficientNetB4, DenseNet201, EfficientNetV2L) 21
5.3 Training the Dense Neural Network (DNN) 23
5.4 Validation and Testing Procedure 26
6 Results 29
6.1 Model Performance Evaluation 29
6.2 Discussion on Results 32
6.3 Respond to Research Questions 34

7 Summary 35
References 37

Figures

Figure 1: Deepfake Detection Pipeline Using CNN Feature Extractors and DNN Classifier 2
Figure 2: GPU Resources on Google Colab Environment 20
Figure 3: GPU Resources on Puhti Supercomputer Environment 20
Figure 4: Visual Deepfake Detection Results Showing Model Predictions on Test Images 23
Figure 5: Training and Validation Metrics – EfficientNetB4 + DNN 25
Figure 6: Training and Validation Metrics – DenseNet201 + DNN 25
Figure 7: Training and Validation Metrics – EfficientNetV2L + DNN 26
Figure 8: Class Distribution of Real and Fake Images Across Training, Validation, and Test Sets 27
Figure 9: Confusion Matrix – EfficientNetB4 + DNN 28
Figure 10: Confusion Matrix – EfficientNetV2L + DNN 28
Figure 11: Classification Report – EfficientNetB4 + DNN 30
Figure 12: Classification Report – DenseNet201 + DNN 31
Figure 13: Classification Report – EfficientNetV2L + DNN 31
Figure 14: Comparative Performance of Deepfake Detection Models Across Evaluation Metrics 33
Figure 15: Test Evaluation Output – EfficientNetB4 + DNN 33
Figure16: Test Evaluation Output – DenseNet201 + DNN 34
Figure 17: Test Evaluation Output – EfficientNetV2L + DNN 34

Tables

Table 1: Comparison of pretrained CNN feature extractors used for deepfake classification. 21
Table 2: Training Configuration Comparison for All Three Models 24
Table 3: strengths and weaknesses of each model across different evaluation metrics. 29
Table 4: Qualitative Comparison of Models Based on Observed Behavior 32

Program codes

Program Code 1: EfficientNetV2L Model Architecture and Data Augmentation in TensorFlow 23

Appendices

Appendix 1: Data Management Plan 40
Appendix 2. Dataset Loading and Preprocessing Code (TensorFlow) 40
Appendix 3. Model Training and Checkpointing Code 41
Appendix 4. Performance Evaluation Code (EfficientNetV2L) 42
Appendix 5. Confusion Matrix Visualization Code (EfficientNetV2L) 43

1 Introduction

The rapid development of artificial intelligence in the last decade has created serious security risks. One of the most significant threats is deepfake technology, which has challenged cybersecurity due to its ability to create fake and convincing images and videos. This phenomenon, which was initially considered a form of entertainment, is now widely used to achieve criminal goals such as fraud, identity theft, and disinformation campaigns.

The study presented here is part of a research project implemented at HAMK University, which aims to investigate the application of artificial intelligence in the field of cybersecurity, specifically for the detection of synthetic and fake visual content that increasingly threatens individuals and organizations. Alongside theoretical development, the main objective of this research is to evaluate the practical value of this system in real-world applications such as identity verification, online banking, and video conferencing platforms.

This research designs and evaluates a deepfake detection system using computer vision and deep learning techniques. The system proposed here uses EfficientNetB4, DenseNet201, and EfficientNetV2L for feature extraction and Dense Neural Network (DNN) for binary classification of real or fake visual content. The model is primarily designed for use in real-world cybersecurity environments, especially for identity verification and digital communication platforms.

The research questions that underlie the present research are posed as follows:

- How can deepfake detection models be optimized to improve accuracy while maintaining real-time performance?
- What are the most effective computer vision techniques for identifying manipulated media in cybersecurity applications?
- How can AI-driven deepfake detection be integrated into existing cybersecurity frameworks to enhance protection?

This thesis follows a straightforward structure. It begins with the fundamental theories and concepts of deepfakes, computer vision, and cybersecurity. It is followed by a description of the research process, data used, and prototype development process. The following chapter has the findings, which lead to recommendations for use and suggestions for future

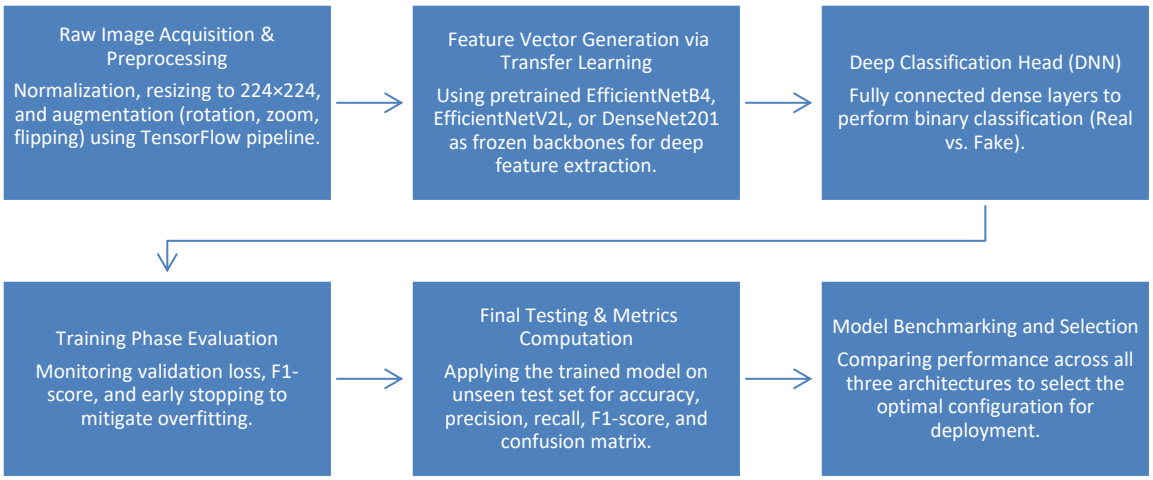
work.

2 Theoretical Foundations and Conceptual Framework

Against the backdrop of the quickly changing challenges presented by AI-generated media, most specifically deepfakes, it is essential to understand basic concepts of deep learning, computer vision, and cybersecurity(Chesney & Citron, 2018, p. 147). The theory outlined in this chapter forms the basis for the empirical part of this thesis, which entails a critical review of existing literature, technologies, and key concepts that pertain to developing a system for deepfake identification.

This diagram summarizes the pipeline used in this thesis to detect manipulated media using pre-trained CNNs and DNN-based classification.

Figure 1: Deepfake Detection Pipeline Using CNN Feature Extractors and DNN Classifier



Chapter 2 is divided into four interrelated subsections. Subsection 2.1 is an overview of different deepfake detection models, including both traditional and deep learning methods. Subsection 2.2 highlights the critical role played by Convolutional Neural Networks (CNNs) to extract image features. Subsection 2.3 discusses how data mining and deep learning methods converged to find applications within cybersecurity, highlighting how jointly, the

methods contribute to detecting sophisticated threats. The integration of these methods was found to improve intrusion detection systems' accuracy and efficiency by applying large datasets to recognize sophisticated patterns and anomalies(Fattahi, 2024, p. 29_35). Finally, Subsection 2.4 elaborates upon deep learning architectures like EfficientNet, explaining why these have been chosen for utilization by the current study.

2.1 Deepfake Detection Models

Rapid development of artificial intelligence, along with deep generative models, has significantly enhanced deepfake content creation, thus elevating concerns about misinformation, identity fraud, and national security issues. Consequently, there is a dire need for advanced and efficient detection methods. Detection methods for deepfakes are generally categorized into two major categories: classical methods based on manually crafted features and modern methods based on deep learning, which have the capability to effectively identify sophisticated visual patterns learned through data (Tolosana et al., 2020, p. 4).

Scholarly research targets deep-learning architectures which have proven to have a capability to recognize sophisticated patterns within image datasets. Specifically, two frameworks have been utilized:

Dense Neural Network (DNN)

Once Detection Models implements the feature extraction function, a Dense Neural Network is trained to perform the task of classifying and labeling images as “real” or “fake”. The neural network consists of multiple dense layers and ReLU activation functions, ending with an output layer that uses a sigmoid function. The training process is conducted by either the Adam optimizer or Stochastic Gradient Descent (SGD) (Goodfellow et al., 2016, p. 198_205). The latter two steps have significant advantages, such as reducing the data size, shortening the training time, and improving the required accuracy (Afchar et al., 2018, p. 5).

2.2 Convolutional Neural Networks and Feature Extraction

Among the image processing and visual analysis models, convolutional neural networks (CNNs) have been one of the most successful. Convolutional neural networks model the

features and functions of the human visual cortex and can implement the function of learning the automatic spatial hierarchy of image features with layered representations well (LeCun et al., 2015a, p. 438). Traditional methods have been based more on human-engineered features, but CNNs can simultaneously identify meaningful patterns in raw visual data, which facilitates its application in the real world, including face recognition, image classification, and deepfake detection, especially the latter requiring the detection of subtle and highly complex artifacts (Rawat & Wang, 2017, p. 4).

Convolutional neural networks have a traditional architecture that consists of different layers: convolutional layers, ReLU activation functions, pooling layers, and fully connected layers as the culmination. First, there are initial convolutional layers designed to identify low-level features, like textures and edges, and later, subsequent layers to identify higher-level and complex features, including facial features or subtle differences between original and manipulated content(Albawi et al., 2017, p. 2).

In this study, three state-of-the-art convolutional neural networks EfficientNetB4, DenseNet201, and EfficientNetV2L were used for high quality feature extraction. These architectures, which are designed to optimize accuracy while maintaining computational efficiency, can significantly reduce the computational cost and number of parameters while maintaining the accuracy of the model, which is especially useful in cases where there is a need for real-time inference or deployment in resource-constrained environments (Sandler et al., 2018, p. 4).

2.3 Data Mining and Deep Learning in Cybersecurity

Data mining refers to the process of extracting meaningful patterns, trends, and observations from large data sets. Data mining is a significant process in cybersecurity that identifies anomalies, assesses active attacks, and can predict impending threats. These algorithms use logs, network traffic, and user actions to generate alerts and mitigate threats (Chandola et al., 2009, p. 5_6).

Traditional data mining has limitations that have been overcome by recent advances in deep learning. New methods such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) can evaluate complex data sets without the need for traditional data mining methods, and it is these features that allow these deep neural networks to detect new complex threats through subtle patterns. (Kim et al., 2014, p. 3_4).

Deepfake detection can combine deep learning techniques and data mining methods, thereby enhancing the ability to detect changes immediately in visual content (images and videos). EfficientNet, XceptionNet, and MobileNet can function as good features extractors, providing rich vector representations for use within classification models, including DNNs and SVMs, and thus enabling accurate prediction(Nguyen et al., 2019, p. 3).

2.4 Deepfake Generation Techniques

Deepfake technology draws upon sophisticated forms of artificial intelligence which can generate remarkably realistic audio-visual content, mimicking a subject's facial expressions, voice features, and motions. The mass appearance of these fake media has increased significantly as there have been improvements in machine learning and computer vision methodologies. The speed with which this technology advances is, for the most part, attributed to readily available strong generating models as well as large sets of visual data contained on the world wide web and used to feed the algorithms (Westerlund, 2019, p. 1_2)

Among all the methods used for deepfake creation, Generative Adversarial Networks (GANs) hold a vital place. GANs contain two different kinds of neural networks: a generator that creates fake content and a discriminator that distinguishes real and created content. Through this adversarial process, it is possible to continually improve the generator to produce realistic results (Goodfellow et al., 2014, p. 1_2) . Moreover, Autoencoders models of unsupervised neural networks are used worldwide, mainly for facial reconstructions. These networks process visual features by reducing them and then rebuilding attributes, which makes them good at altering facial features and changing identities (Kingma & Welling, 2022, p. 5).

A key methodology used under these circumstances is face swapping, which involves incorporating a source identity into a target video via facial landmarks and alignment methods. Unlike GANs and autoencoders based on generative learning methods, face swapping occasionally relies on traditional methods of computer vision or hybrid models. While these technologies originally have entertainment purposes, they have been adapted for other purposes like disseminating disinformation, impersonation, and cybercrime, thus posing serious cybersecurity and personal privacy risks (Kietzmann et al., 2020, p. 7) . These production methods can be used to develop more powerful deepfake detection systems.

2.4.1 Generative Adversarial Networks (GANs)

One of the current significant frameworks widely used for creating deepfake content is Generative Adversarial Networks (GANs). According to Goodfellow et al. (2014), GANs often have a rival neural network: 1. A generator that produces fake data such as images of faces or fake video frames, and 2. A discriminator that can confirm or reject the similarity of the data as real or fake. Both models are trained simultaneously using a minimax framework. In this process, the generator aims to mislead the discriminator, and the discriminator aims to improve the detection of imitations. Ultimately, the system output can be fake but realistic images and videos.

GANs have recently made great progress in facial synthesis, image-to-image translation, and video manipulation. GANs help style transfer technologies to create realistic and fake composite faces or swap facial identities in videos (Karras et al., 2019, p. 1_2). Models such as StyleGAN and DeepFaceLab are now widely used to modify facial expressions, lighting conditions, and age changes with very high accuracy. With their high ability to assimilate and learn the distribution of complex data based on real-world samples, these technologies can generate content that is almost impossible to identify without advanced analytics techniques.

Although GANs are technologically and innovatively sophisticated, they have also raised serious security and ethical issues in today's world, including their use to generate fake speeches, create images without people's consent, and online scams, the undesirable results of which demonstrate the need for tools such as frequency analysis, detection of temporal inconsistencies, and adversarial training techniques to detect them (Nguyen et al., 2022, p. 6). A thorough understanding of how GANs function is essential for developing effective detection systems that can address this escalating cybersecurity threat.

2.4.2 Autoencoders

Autoencoders are essentially unsupervised neural networks that first encode data and then rebuild the data by mapping it to a low-dimensional space and finally returning it to its original state. Such a network typically consists of two major components: an encoder, which maps the input (for example, facial image) into a latent representation, and a decoder, which tries to restore input based upon this compressed state. Autoencoders have proven to be efficient at extracting and encoding inherent characteristics of facial features while, at the same time, filtering out noise or unnecessary variability (Hinton &

Salakhutdinov, 2006, p. 504). Their ability to learn representations tailored to specific persons' identities made them an essential tool used in many early deployments of deepfake technology.

Autoencoders have traditionally been trained with two different identities, utilizing a single shared encoder with two separate decoders for both source and target facial representations. In inference, a source facial representation is input to the shared encoder, reconstructed by the target's decoder, and hence seamlessly transmitting expressions and structural features of the source to the target's facial appearance. Such a methodology allows realistic swapping of faces at relatively low data and computational requirements(Nguyen et al., 2019, p. 3). Albeit newer and more sophisticated methods like Generative Adversarial Networks (GANs) have surpassed autoencoders with regards to realism of output, autoencoders remain a foundational framework for understanding and generating facial manipulations and are still used for lightweight or hybrid systems for generating deepfakes.

2.4.3 Face Swapping

Face swapping is a technique commonly used for generating deepfakes, which refers to an operation of substituting one's facial features for those of another in an image. This process initially requires facial landmarks prominent reference points (such as eyes, nose, and mouth) to align the source and target faces. After aligning faces, facial features of sources are adjusted to meet structural elements of the target's face, ensuring coherence in pose and expression. Advanced methods involve technologies like Delaunay triangulation for large deformations and Poisson image editing for smooth blending, leading to realistic composites(Xu et al., 2022, p. 3).

Substantial advances in technology have made it easier for individuals with limited technical expertise to access and utilize deepfake tools. Various open-source software, such as DeepFaceLab and FaceSwap, rely on advanced algorithms for face recognition, alignment, and content quality enhancement to enable face swapping without requiring the user to have technical knowledge. What has been a growing concern is the facilitation of fake content generation through these algorithms, which can be used to achieve malicious goals and pose threats to individuals and organizations. Consequently, the development and strengthening of fake content detection systems and algorithms is essential to mitigate the risks and concerns related to face swapping (Nirkin et al., 2022, p. 3).

3 Deep Learning and Computer Vision in Deepfake Detection

The development of computer vision and its recent achievements, especially when combined with deep learning techniques, have played a pivotal role in the development of systems that can detect deepfakes. Deep neural networks now allow for the automatic recognition of complex patterns in visual data. In particular, these capabilities help to distinguish between authentic visual content and fake or synthetic content (LeCun et al., 2015b).

Computer vision as a subfield of artificial intelligence allows machines to process and interpret complex visual information contained in images and videos. By combining computer vision with advanced deep learning methods, a powerful tool can be achieved that facilitates and enhances the ability to identify visual anomalies such as irregular facial expressions, inconsistent lighting, and abnormal eye movements (Zhou et al., 2017, p. 3).

Convolutional Neural Networks (CNNs), EfficientNet, and Vision Transformers are now being considered to improve the performance of existing deepfake detection models. Similarly, transfer learning and data augmentation techniques can be used to address the limitations of insufficient labeled data, accelerate training, and increase model accuracy (Tan & Le, 2020, p. 2).

3.1 Convolutional Neural Networks (CNNs) in Deepfake Detection

Among deep learning methods, CNNs are one of the most significant and powerful methods for image analysis and classification. These algorithms, which attempt to mimic the architecture of the human visual cortex, can learn hierarchical representations of spatially based characteristics and handle visual information such as facial images (Goodfellow et al., 2016; LeCun et al., 2015a). The common framework of these algorithms has certain characteristics: 1. Convolutional layers that extract features, 2. Activation functions such as ReLU bring in non-linearities, downsampling layers, and fully connected layers that generate final predictions.

3.1.1 Pretrained CNN models to identify deepfake

CNNs can detect deepfakes by considering the smallest visual differences that the naked eye cannot detect, such as unnatural skin texture, uneven lighting conditions, and facial

imbalance (Nguyen et al., 2019, p. 3). This unique capability is enhanced by pre-trained models as feature extractors (e.g., MobileNetV2, EfficientNetB4, and EfficientNetV2L). Pre-trained models are trained using large datasets (such as ImageNet) and can use pre-learned feature representations that require minimal training periods (Howard et al., 2017, p. 2; Pan & Yang, 2010).

Among these architectures, MobileNetV2 is more popular because it has features such as lightweight and fast processing, which makes it more effective for use in real-time applications or deployment in resource-constrained environments. EfficientNet models, on the other hand, obtain a balance between computational efficiency and high accuracy by scaling the depth, width, and resolution uniformly (Tan & Le, 2020, p. 5).MobileNet and EfficientNet are some of the lightweight convolutional neural networks that have proved their efficiency in visual feature extraction while also lowering the computational requirements (Mohsin, 2023, p. 14).

3.1.2 Comparative performance of CNNs

Comparative studies between several CNN models like VGGNet, ResNet, MobileNetV2, EfficientNetB4, and EfficientNetV2L have established varied trade-offs between processing speed and accuracy. While models like MobileNetV2 are highly desirable in cases of fast processing, these are likely to yield low performance on complex datasets compared to more robust models like EfficientNetV2L or XceptionNet (Rössler et al., 2019, p. 5). These deeper networks have the capacity to recognize more detailed spatial patterns and visual differences, crucial in distinguishing deepfakes from real pictures.

3.2 Optimization Models and Transfer Learning

Transfer learning is a key approach to deep learning, especially where labeled data is scarce or prohibitively expensive. Instead of training a model afresh, transfer learning allows for applying what is learned from a different task to address a similar task. It generally entails applying a pre-trained model obtained from a large dataset, for example, ImageNet, and fine-tuning higher-level layers to adapt to a specific task at hand. Such an approach not only reduces computational costs and training time but also improves performance, as illustrated through deepfake detection, which requires careful inspection of sophisticated image features (Pan & Yang, 2010).

Transfer learning is crucial to the field of deepfake detection since it endows models with the ability to distinguish fine and complex differences within visual data. General image classification frameworks that have been pre-trained, like EfficientNetV2L and EfficientNetB4 originally intended for larger image classification tasks have proven to be effective as feature extractors. The early layers of these architectures learn low-level visual features, like contours, textures, and edges, and later layers learn advanced and abstract features specific to the targeted task (Tan & Le, 2020, p. 7) . Hierarchical representations of features are crucial for distinguishing between original and manipulated facial images.

3.3 The Role of Computer Vision and Feature Extraction

Computer Vision is an area of study under the general umbrella of artificial intelligence that is concerned with developing systems capable of interpreting, analyzing, and understanding visual information found within static images and video streams. Deep learning methods have been major drivers of developments across a variety of fields, including facial recognition, image classification, and detecting changes (for example, changed, forged, or reproduced products) (Szeliski, 2022, p. 333)

Feature extraction seeks to define key information found in images that can be used as inputs for various machine learning applications. The feature categories can represent a wide range of elements, including edge structures, texture, and color gradients, as well as more complex indicators, such as abnormal facial expressions or unique skin pattern(Goodfellow et al., 2016, p. 326).

A common technique for extracting features in the field of computer vision is through applying Convolutional Neural Networks (CNNs). For purposes of this project, EfficientNetB4, DenseNet201, and EfficientNetV2L were used as specified feature extractors. These models, trained using ImageNet for feature learning, have been shown to be effective at extracting rich and deep features from complex images while maintaining a balance between accuracy and computational cost (Sandler et al., 2018, p. 5). The features are first extracted by this model and then converted into numerical vectors, and a Dense Neural Network (DNN) is used to confirm or reject the image classification as forged.

3.4 Overall Framework of a Deepfake Detection System

A multi-level process, which has multiple steps to analyze and classify video or image content, is implemented for fake content detection; for example, data collection, preprocessing, feature extraction, classification, and final decision-making (Tolosana et al., 2020, p. 1_2). The research presented here suggests an advanced version of this model, which has different steps:

Image Preprocessing

Data preprocessing plays a pivotal role in preparing the dataset, especially before training and model evaluation can be performed. Preprocessing typically involves various methods, including transformation of color space, where pictures are transformed from Blue-Green-Red (BGR) to RGB or, under specific cases, to grayscales to provide input formats with consistency. Pixel normalization is another important process, which includes rescaling pixel values often between 0 and 1 towards ensuring numerical stability during model training and accelerating optimization algorithm convergence rates (Shorten & Khoshgoftaar, 2019, p. 8)

Along with the critical initial procedures, an important preprocess technique is data augmentation, which is designed to increase both the size and variety of the training data artificially. Commonly used augmentation methods include random horizontal or vertical flips, rotations, zooming, and changes to contrast, all of which reproduce different conditions of actual images and enable more generalization by the model. These methods are especially valuable for use with relatively small datasets since they protect against overfitting, a situation where the model memorizes training data verbatim rather than learning more generally applicable attributes (Shorten & Khoshgoftaar, 2019, p. 9)

Feature Extraction Using Pre-Trained Models

Convolutional neural network models, for example, EfficientNetB4 and MobileNetV2, are currently used for visual data feature extraction. These models generally recognize basic patterns, i.e., edges, in early layers, later moving to recognizing complex structures in the later layers (Tan & Le, 2020, p. 4). EfficientNetV2L, as a more recent and optimized architecture, offers improved training speed and accuracy trade-offs, making it particularly suitable for large-scale deep-fake detection tasks.

Classification Using a Dense Neural Network (DNN)

The extracted features are then fed into a DNN, which uses dense layers with ReLU and sigmoid activation functions to perform the final classification of whether the image is real or fake. After classifying the data, the model is evaluated using the test dataset and metrics such as accuracy, recall, F1-score, and confusion matrix that can provide an adequate understanding of its performance (specifically imbalanced classification tasks) (Saito & Rehmsmeier, 2015, p. 9). At the same time, the training performance is evaluated using the analysis of accuracy and loss curves at different periods, which can provide a reasonable understanding of the model.

3.5 Common Challenges in Deepfake Detection

Although deep learning and computer vision models capable of detecting fake content have advanced greatly in recent years, there are challenges that hinder the development of the accuracy of these models (Tolosana et al., 2020, p. 5)

Lower Quality Video or Image Sources

Many fake videos are typically highly compressed or have low resolution, which affects the performance of fake content detection algorithms and hides subtle cues of authenticity, such as micro-modes or inconsistent lighting. Consequently, there is a need to develop fake content detection techniques that can detect corrupted visual input (Verdoliva, 2020, p. 12).

Rapid Advancements in Deepfake Generation

As fake content detection technologies evolve, so do deepfake generation techniques. For example, GANs have become much more sophisticated in recent years and can generate synthetic content that current fake content detection systems simply cannot identify. As a result, the numerous attempts to generate fake content and systems that detect deepfakes indicate that the continued development of new detection techniques is of great significance (Tolosana et al., 2020, p. 4).

4 Methods

This chapter discusses the implementation and testing of deep learning models for deepfake detection and provides a general description of the framework, datasets, libraries, and tools used in the implementation, details of how the models were designed, and metrics that evaluate the performance and effectiveness of the models.

This research deals with three different models and a comparative and quantitative analysis. First, a feature extractor, here EfficientNetB4, and a DNN as a classifier are used. The second model is built based on DenseNet201, which is paired with a DNN. Finally, in the third model, EfficientNetV2L is combined with a DNN and implemented. The experiment was conducted on all three models under standardized experimental conditions such as a similar dataset, batch size, number of epochs, optimizer configuration, and evaluation techniques to have a stable and consistent platform to compare experimental results.

This methodology attempts to systematically implement and evaluate the performance and contribute to the broader field of fake content detection (here, deepfake) by examining the advantages and disadvantages of different deep learning paradigms. At the same time, it is expected that the findings of this study will help to strengthen and improve cybersecurity measures regarding manipulated content.

4.1 Research Design

In this research, a systematic experimental design is implemented to evaluate the quality and capabilities of deep learning techniques for detecting fake content (here, deepfake) in several steps: 1. data preprocessing, 2. feature extraction using pre-trained CNNs, and 3. classification with specially crafted models. The main objective of this systematic design is to create a reliable deepfake detection platform and simultaneously compare it with other deep learning models.

To test and maintain the stable training of the model, the dataset was divided into training, testing, and validation subsets. The training data was mainly used to optimize the parameters and the validation data was used to improve the model and implement early stopping techniques to avoid overfitting. Meanwhile, the last test set was dedicated to testing the performance of the model.

The present study implements three main architectures for deepfake detection: 1. Feature extraction via EfficientNetB4 alongside a DNN for data classification; 2. Taking care of the data classification task using DenseNet201 and a DNN; and 3. Feature extraction and classification via EfficientNetV2L and DNN. For a balanced and unbiased comparison, all three models were tested and evaluated in terms of dataset distribution, batch size, number of epochs, and optimization parameters.

To assess the effectiveness and reliability of the models, performance metrics such as accuracy, loss, confusion matrix, and classification report were collected and analyzed. This structure also allowed real-world implementation of deepfake systems along with allowing a comprehensive comparative analysis between models, leading to strong conclusions about their strengths and weaknesses.

4.2 Dataset Description

A set of images of faces is selected as data, which are classified into two categories: real and deepfake. This set of images is collected from public databases. The selected images are carefully checked to ensure their consistency and quality for building a deep-learning model-based classifier. The goal of collecting these images was to obtain a dataset that shows the original and manipulated facial content.

The aforementioned dataset is classified into three non-overlapping subsets: 1. a training dataset containing 17,920 images for model optimization, 2. a validation dataset containing 7,680 images for hyperparameter tuning and in early stopping implementation to avoid overfitting, and 3. a testing dataset containing 6,400 images for final performance evaluation. This division allows for the development of a systematic evaluation strategy.

All images were resized and saved at 224 x 224 pixels; in addition, the data was normalized to meet the input requirements of the deep models. At the same time, several data augmentation processes were performed (random flipping, rotation, zooming, and contrast adjustments) to increase model generalization and prevent potential overfitting.

The data used in this study was systematically labeled into multiple folders in a manner corresponding to individual subsets and classes to improve the efficiency of data loading through TensorFlow pipelines. In other words, an optimized and repeatable process was adopted for the model training and validation steps, based on EfficientNetB4, DenseNet201, and a mix of EfficientNetV2L with DNNs.

4.3 Tools and Libraries Used

The model implemented in this research to detect deepfakes uses the Python programming language, which has the necessary capabilities for tasks related to machine learning and collision image processing. The implementation applied several open-source tools and libraries whose selection was based on their applicability and performance throughout different steps encountered in a deep learning process.

TensorFlow and its main application programming Keras are the main tools for model development and training. These tools provide built-in functions that allow importing and fine-tuning existing models (EfficientNetB4 and DenseNet121), which facilitates the design of complex neural networks.

Data manipulation was performed using NumPy and Pandas, more specifically with respect to handling labels, computation of dataset statistics, and operations on intermediate data structures in the preprocessing and evaluation steps. The OpenCV library was applied for image-related operations such as resizing, reading, and format conversion.

To visualize performance metrics and trends throughout training and testing phases, Matplotlib and Seaborn were used to generate plots including accuracy/loss curves and confusion matrices.

The overall coding and development process was carried out in a Jupyter Notebook context, which supported modular and interactive progress. The availability of code cells, in-line visualization, and debugging tools made Jupyter especially suitable for iterative testing and evaluation.

Datasets and models were retrieved and saved through Google Drive to easily integrate future cloud-based services like Google Colab when necessary.

In short, integration of these libraries and tools enabled an extremely efficient, transparent, and reproducible implementation of the deepfake detection framework.

4.4 Model Architecture and Configuration

The proposed deepfake detection framework in this work involves three major architectural models: the first model has a customized architecture with EfficientNetB4 acting as the feature extraction block followed by a Dense Neural Network (DNN) as the classifier; the second model has a combination of DenseNet201 followed by a Dense Neural Network (DNN); and the third model utilizes EfficientNetV2L combined with a Dense Neural Network (DNN).

In the first model, EfficientNetB4 was used as the feature extraction backbone of the face images. The EfficientNetB4 network was pre-trained on ImageNet weights and was used without its topmost classification layer in its original form as a feature extractor. The features were then fed into a Dense Neural Network composed of fully connected layers with ReLU activation functions and regularization through dropout. The final classification layer used a sigmoid activation function to generate a binary label representing an authentic or a forgery input.

The next model was built based on the framework of DenseNet201 and was initialized with weights trained on ImageNet and adapted through removal of its original classification module. A new classification similar to the one used in the EfficientNetB4-based model is proposed to ensure architectural consistency and a balanced representation.

The third model is based on EfficientNetV2L, a new architecture that initiates with pre-trained weights and extends with a DNN to allow binary classification.

The three models mentioned were generated through the Adam optimization algorithm with a learning rate of 0.0001 and a binary cross-entropy loss function. The training process was implemented for several epochs with early stopping and model-checking methods to control overfitting and maintain the best model.

This design allows for the creation of a model with the characteristics of efficiency, flexibility, and robustness by combining the strengths of pre-trained convolutional neural networks with classifier layers, enabling deepfake detection. Finally, the models were tested on a test set and their performance was evaluated through the collected classification metrics to confirm comprehensive testing.

4.5 Evaluation Metrics

Several standard metrics were implemented to evaluate the performance of models designed to detect deepfake, which allows for a complete understanding of the performance of each model in distinguishing between original and manipulated data at different testing steps.

The main criterion for evaluating the models in this study is accuracy; in other words, the overall performance measure, which is expressed as the ratio between correctly classified images of valid and fake samples and the total test samples. Accuracy is an effective measure for evaluating overall performance, but it is wanting in cases where there is class imbalance and a difference in significance between false positives and false negatives.

At the same time, the study also examined additional measures such as precision, recall, and F1 score. Precision assesses how well positive predictions are made in relation to the total number of positive predictions. Recall indicates how well a model can detect all deepfakes in a dataset. The F1 score, as a harmonic mean between precision and recall, is a measure of balance when both measures are correlated.

The performance of the interventions was estimated against the classification_report function in Scikit-learn based on detailed analyses of the test dataset. The outcomes were expressed in numeric form in addition to visual aids through the use of bar charts to readily compare the three models: EfficientNetB4 + DNN, DenseNet201 + DNN, and EfficientNetV2L + DNN. The visual aids were significant in concluding the comparative strengths and weaknesses recognized in each design process.

The use of these evaluative criteria in this study guarantees a reliable, transparent, and systematic analysis of deepfake detection tool usefulness, thus improving the credibility and stability of the findings acquired.

5 Practical Implementation

This step represents the methodology used in building, training, and testing models geared towards deep-fake detection. This phase outlines in-depth studies of methodologies used in system design, data preprocessing, and modeling in various strategies. The aim in this

phase was to translate into a working prototype a conceptual outline capable of distinguishing between real and deepfaked images.

The practical implementation began with setting up the coding environment. The first model, EfficientNetB4 combined with a Dense Neural Network (DNN), was trained on Google Colab Pro, to access which a subscription service was provided. While Google Colab Pro provides several strengths, its limits in terms of GPU and capacity made it impossible to progress to training of other, more complex models in that environment. The next two models, including DenseNet201 combined with DNN and EfficientNetV2L combined with DNN, were thus trained in the Puhti supercomputer environment, which provided the necessary computational power to support larger and complex deep learning initiatives.

Following setting of the environment, loading and preprocessing of the dataset involved taking measures such as resizing, normalization, and data augmentation. The models were also faced with similar experimental conditions with early stopping and checkpointing to keep only the best-performing model weights.

The performance was evaluated on a model-by-model basis using the specified test set. The results were examined through several performance visualization methods such as loss and accuracy curves, along with a confusion matrix to test effectiveness in classification. Careful notes were documented and reviewed of each implementation step so as to authenticate the technological feasibility of applying deep learning and computer vision techniques towards differentiating deepfakes.

5.1 System Setup and Environment

The work was initiated with two development environments. The initial one was Google Colab Pro due to its ease of use and access to more powerful GPU resources through a subscription model. However, time and memory constraints in Colab Pro hindered the feasibility of prolonged training sessions for complex models. Therefore, the work was shifted to more powerful infrastructure for more convenient operations.

The training was then conducted on Puhti supercomputer, providing capable computational capacities to facilitate deep learning processes. A local development environment was also established simultaneously in Anaconda with a virtual environment "deepfake_gpu" having TensorFlow, Keras, OpenCV, NumPy, Pandas, Matplotlib, and necessary libraries. The

local machine was installed with a Windows 11 operating system and was equipped with an NVIDIA GTX 1650 Ti GPU and 16GB of RAM and was kept ready for development and testing as and when necessary.

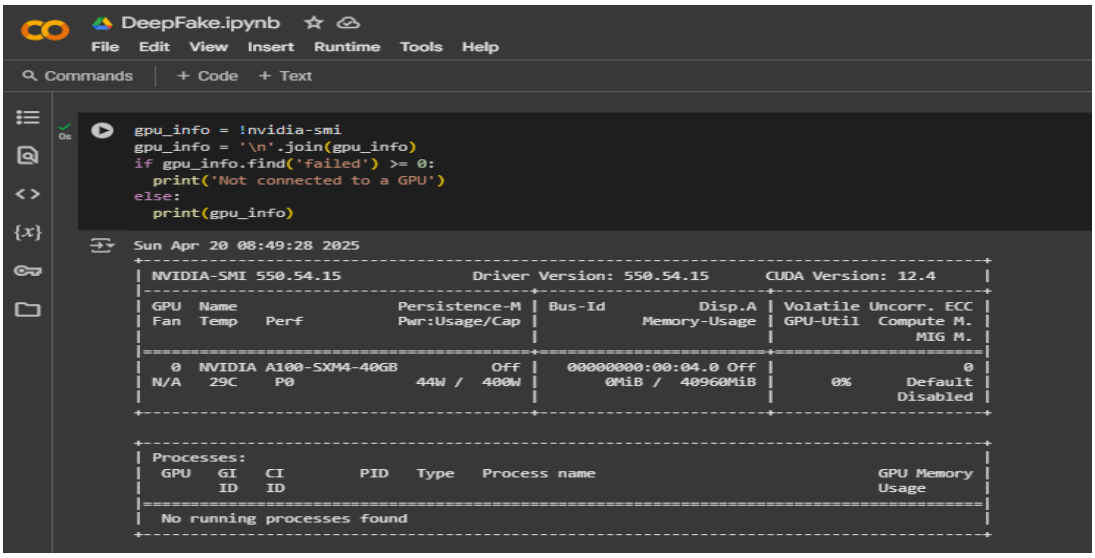
We employed VS Code as our local IDE of choice to allow flexibility in code visualization, structure, and debugging. Well-structured directories were employed to save model checkpoints and data sets, and Google Drive was mounted to Colab sessions to enable cloud storage of models in early training.

This two-environment setup provided flexibility and continuity throughout the course of the project in a manner in which development progressed uninterruptedly even when there were shortages in resources. Seamless transitioning between cloud and high-performance computing environments meant it was easy to execute training and test processes.

Following setting of environment, a dataset was loaded and preprocessed through resizing, normalization, and augmentation. Training was initiated with an EfficientNetB4 model where EfficientNetB4 was employed to serve both as a feature extractor and a Dense Neural Network (DNN) classifier. A model incorporating a DenseNet201 + DNN was also introduced and trained with similar experimentation parameters. Subsequently, a third model based on EfficientNetV2L with a Dense Neural Network was also trained to obtain extra performance gains.

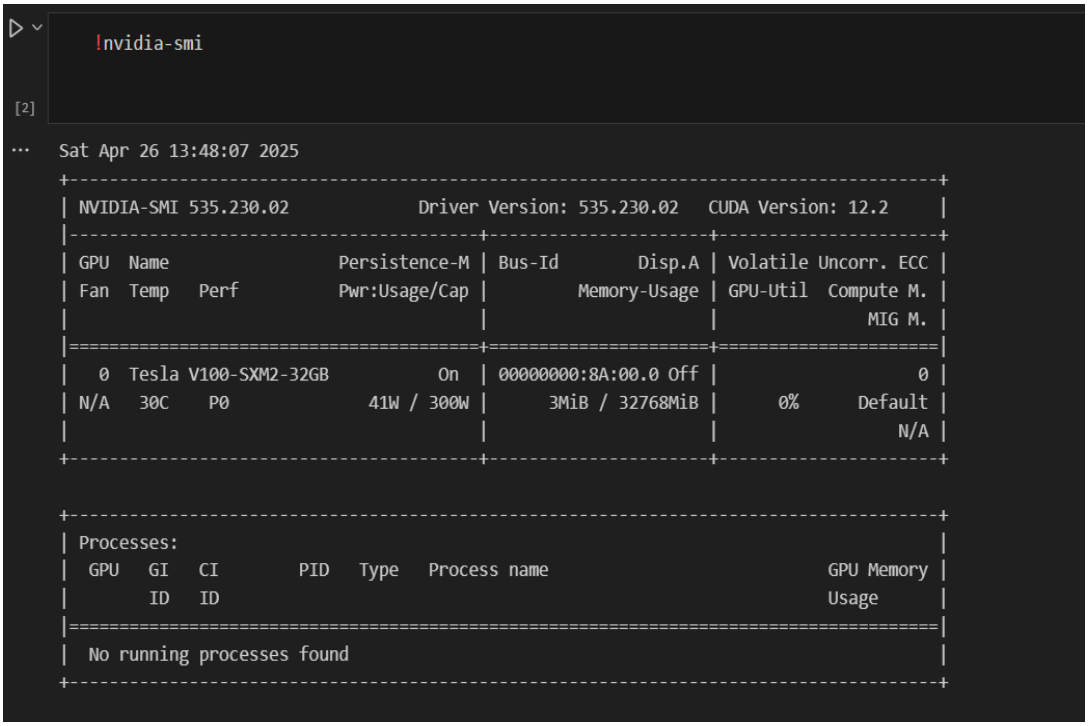
The GPU information is provided in the screenshot taken from Google Colab environment. A Tesla T4 GPU with a memory capacity of 16GB was available for training model one (EfficientNetB4 + DNN). This configuration was sufficient to fully train only model one due to Colab memory and runtime limits.

Figure 2: GPU Resources on Google Colab Environment



Here is an image of GPU specification of our Puhti supercomputer used in training models two and three. We used a Tesla V100-SXM2-32GB GPU with a memory of 32GB and even greater processing capacity. This enabled us to train bigger models (DenseNet201 and EfficientNetV2L) without any memory limitation such as in cloud services.

Figure 3: GPU Resources on Puhti Supercomputer Environment



5.2 Feature Extraction Using Pretrained CNNs (EfficientNetB4, DenseNet201, EfficientNetV2L)

In this study, feature extraction was carried out using three state-of-the-art deep convolutional neural network models: EfficientNetB4, DenseNet201, and EfficientNetV2L. Their selection was based on their known performance in a number of computer vision-related applications and their potential to extract high-quality spatial features out of visual data.

EfficientNetB4 was initially initialized with weights derived from ImageNet, followed by removal of its top classification layers to be used solely as a feature extractor. Similarly, DenseNet201 and EfficientNetV2L were also initialized with pre-trained weights and then fine-tuned to provide only deep feature vectors and without making any final class prediction. The networks were constructed to take input images at a resolution of 224×224 pixels with three channels.

Table 1: Comparison of pretrained CNN feature extractors used for deepfake classification.

Model	Parameters	Input Size	Pretrained On	Output Feature Size
EfficientNetB4	~19M	224×224×3	ImageNet	1792
DenseNet201	~20M	224×224×3	ImageNet	1920
EfficientNetV2L	~121M	224×224×3	ImageNet	1280

Before entering the models, the images were resized and preprocessed according to the assigned preprocessing functions specific to each individual network (for instance, normalization techniques). Data augmentation techniques such as random horizontal flipping, rotation, zooming, and contrast adjustments were also applied to the training data to improve generalizability and prevent overfitting hazards.

The following code snippet shows how EfficientNetV2L can be used as a feature extractor in TensorFlow along with the use of data augmentation layers to improve generalizability in training.

Program Code 1: EfficientNetV2L Model Architecture and Data Augmentation in TensorFlow

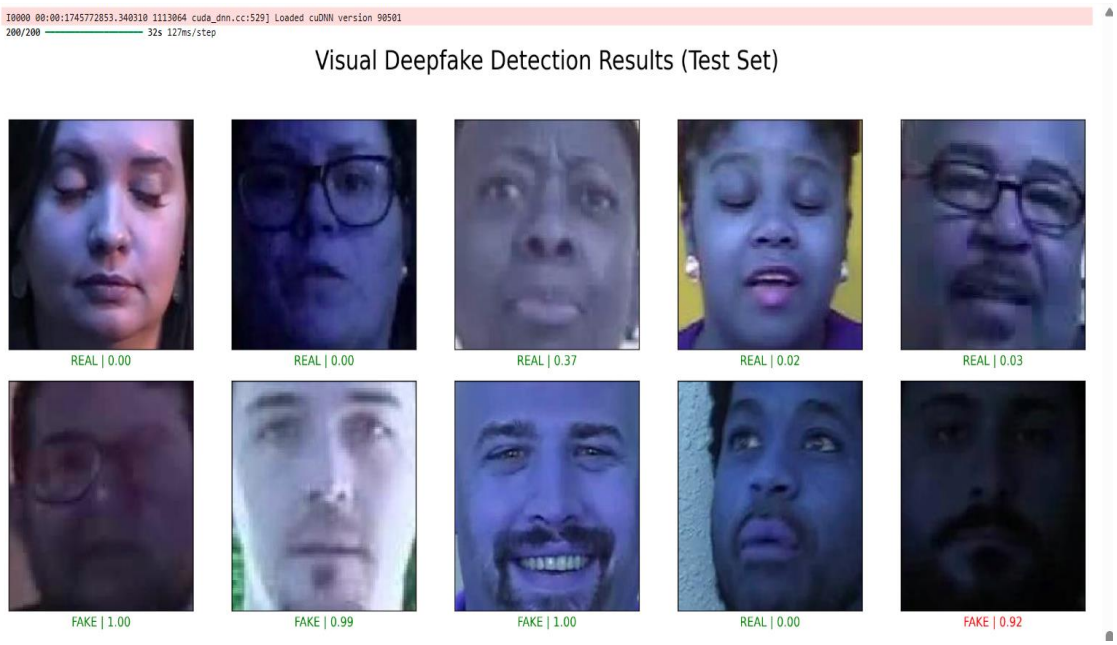
```
# 5.2 - Building the Model Architecture with EfficientNetV2L

from tensorflow.keras.applications import EfficientNetV2L
from tensorflow.keras import layers, models

# Build base model EfficientNetV2L with ImageNet weights
base_model = EfficientNetV2L(include_top=False, weights='imagenet', input_shape=(224, 224, 3))

# Data Augmentation layers
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip("horizontal"),
    tf.keras.layers.RandomRotation(0.1),
    tf.keras.layers.RandomZoom(0.1),
    tf.keras.layers.RandomContrast(0.1),
])

# Build the final model with added layers
inputs = tf.keras.layers.Input(shape=(224, 224, 3))
x = data_augmentation(inputs)
x = base_model(x, training=False)  # inference mode (Base model freezed)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
outputs = tf.keras.layers.Dense(1, activation="sigmoid")(x)

model = tf.keras.Model(inputs, outputs)

# Compile the model with SGD
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
model.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["accuracy"])
```

The feature maps generated by the final sets of convolution blocks in both models were globally pooled and passed through specially designed Dense Neural Networks (DNNs). DNNs were comprised of fully connected layers with ReLU activations, dropout layers for regularization purposes and a final output node with a sigmoid activation.

Using transfer learning of state-of-the-art pre-trained networks, the system would be able to extract rich discriminative features in relation to face images while maintaining minimal training time. The multi-architecture strategy provided a concrete foundation to compare different CNN backbones in a similar experiment setting to reach a good deepfake detection solution.

This is the prediction of the deep-fake-trained model on a test image set. Each image is labeled with predicted class ("REAL" or "FAKE") and corresponding confidence value. Green labels denote correct prediction, and red labels denote incorrect prediction. The results demonstrate how well the model performs in classifying real and manipulated faces even in poor vision conditions.

Figure 4: Visual Deepfake Detection Results Showing Model Predictions on Test Images



5.3 Training the Dense Neural Network (DNN)

Following feature extraction, it proceeded to train specialized Dense Neural Networks (DNNs) appended to each feature extractor design. The DNNs were responsible for distinguishing between extracted feature representation as real or artificial. Training was specifically designed to strike a balance between performance and computational expense.

All DNN comprised an input layer of equal dimensions to feature vectors extracted from corresponding pre-trained CNN models (EfficientNetB4, DenseNet201, and EfficientNetV2L), two fully connected layers with ReLU activation and dropout layers to avert overfitting. The final classification layer utilized a sigmoid activation to classify images into authentic and manipulated.

Training was performed with Adam optimizer and a learning rate of 0.0001 along with binary cross-entropy loss. Models were individually trained on dataset developed with approximately 140,000 images (real: 70,000 and fake: 70,000), for 47 epochs with a batch of 64. Early stopping was applied to prevent training if validation accuracy was not improving, and best models were saved through the feature called ModelCheckpoint.

The training was divided between two platforms. The first model (EfficientNetB4 + DNN) was trained on Google Colab Pro using its available GPU capabilities, whereas training the

other two models (DenseNet201 + DNN and EfficientNetV2L + DNN) in the Puhti supercomputer cluster to perform bigger computations.

The below table shows the major training parameters used in all three models used in this work. The parameters are the training platform, number of epochs, batch size, learning rate, and optimizer with corresponding CNN-based model. The comparison clearly depicts uniform experimental setup used to ensure fairness in comparison between models.

Table 2: Training Configuration Comparison for All Three Models

Model Architecture	Optimizer	Learning Rate	Batch Size	Epochs	Training Platform
EfficientNetB4 + DNN	Adam	0.0001	64	47	Google Colab Pro
DenseNet201 + DNN	Adam	0.0001	64	47	Puhti Supercomputer
EfficientNetV2L + DNN	Adam	0.0001	64	47	Puhti Supercomputer

Model performance was monitored throughout training through plotting training loss and validation loss and accuracy against epochs, and this provided good measures of convergence performance and overfitting. The three models showed good generalization performance on test data with test accuracy values greater than 91% in best cases.

Validation and train accuracy and loss curves of proposed model suggestions EfficientNetB4 + DNN, DenseNet201 + DNN, and EfficientNetV2L + DNN are shown below. EfficientNet models have good stable convergence along with accuracy. DenseNet201 overfits with good train accuracy but poor validation performance.

Figure 5: Training and Validation Metrics – EfficientNetB4 + DNN

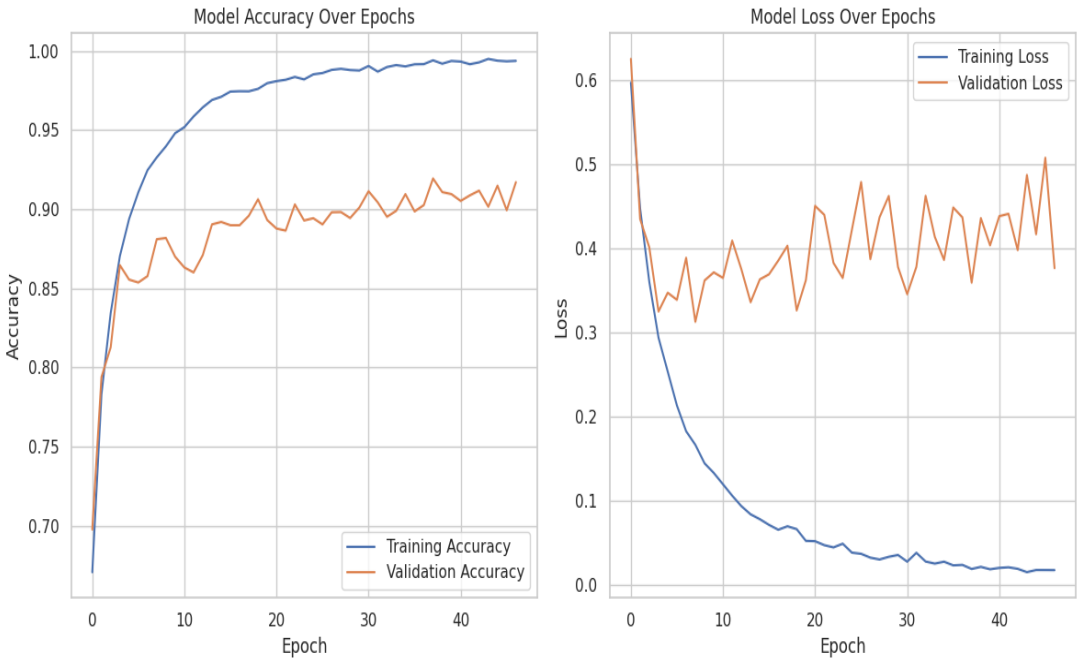


Figure 6: Training and Validation Metrics – DenseNet201 + DNN

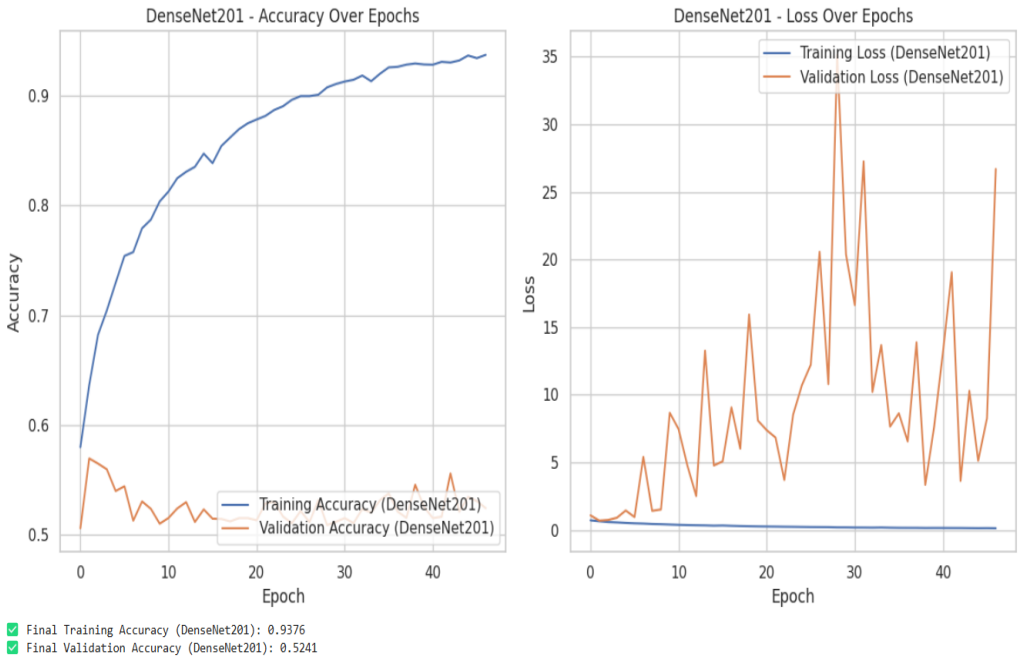
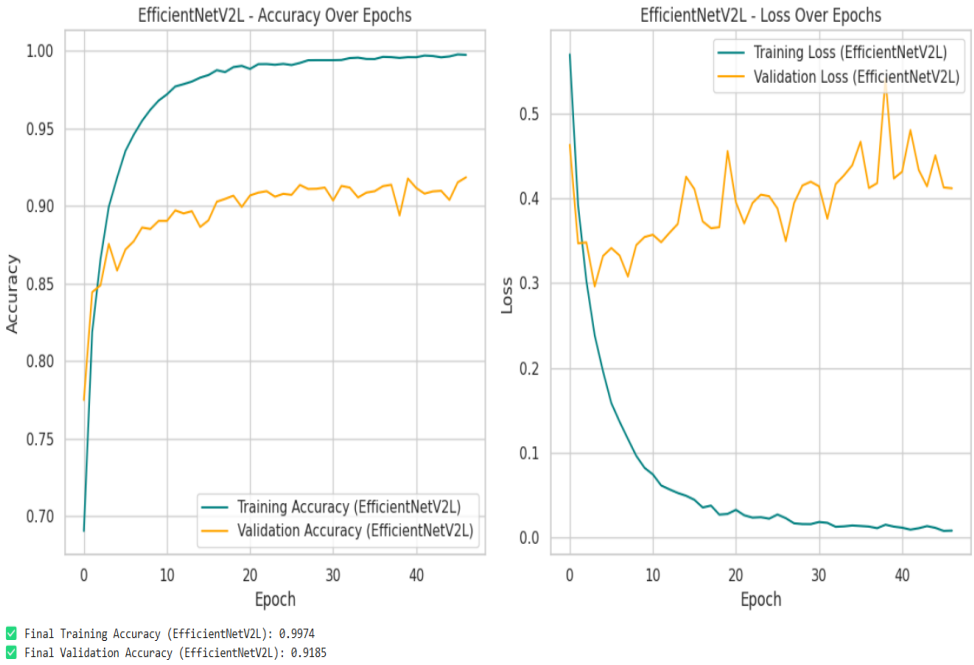


Figure 7: Training and Validation Metrics – EfficientNetV2L + DNN

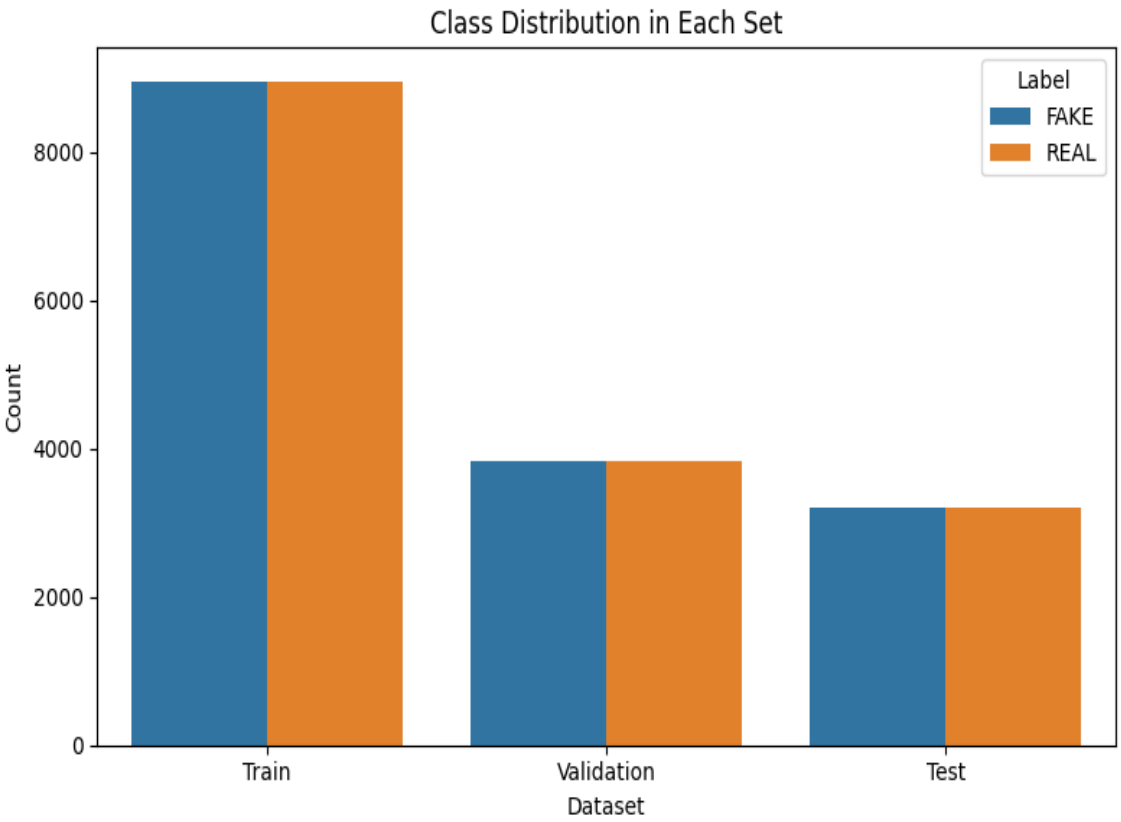


5.4 Validation and Testing Procedure

There was a phase of systematic testing and validation afterwards to evaluate performance in deepfake detection systems following training of DNNs with EfficientNetB4 and DenseNet201 and EfficientNetV2L feature extractors. The purpose of this phase was to ensure model prediction was stable and generalizing well to data never seen before.

The data was divided into training (around 70%), validation (around 20%), and test (around 10%) sets. The validation set was used to monitor models' generalizing performance while training. During training, early stopping was used to halt training when validating loss was no longer improving in a bid to avoid overfitting.

Figure 8: Class Distribution of Real and Fake Images Across Training, Validation, and Test Sets



In final analysis, the models were tested with a standalone test dataset of more than 10,000 images, with a similar representation of real samples and forged ones. Quantitative measures such as accuracy, precision, recall, and F1-score were used to measure each model's classification performance. The results were represented through performance curves, bar charts, and confusion matrices to enable detailed analysis.

The following confusion matrices show the results of classifying each model when run against the test set, clearly representing the distribution of true positives, true negatives, false positives, and false negatives.

Figure 9: Confusion Matrix – EfficientNetB4 + DNN

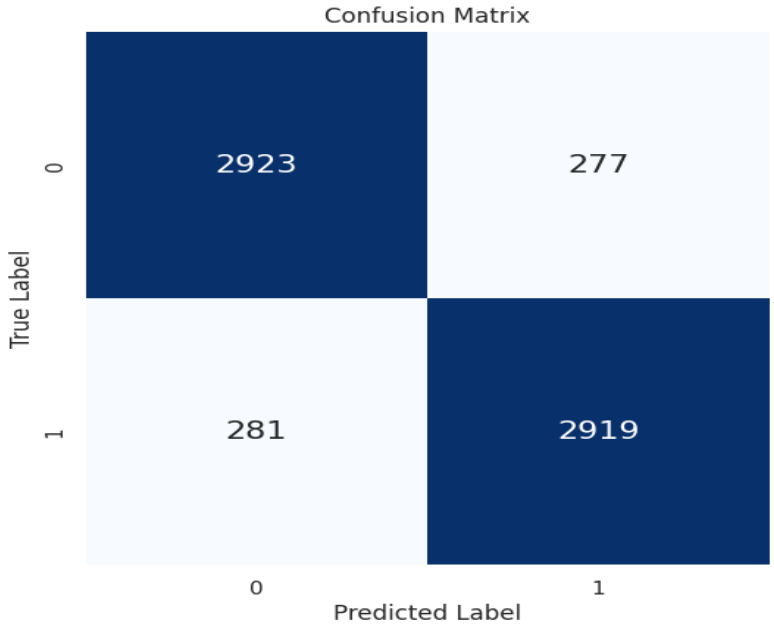
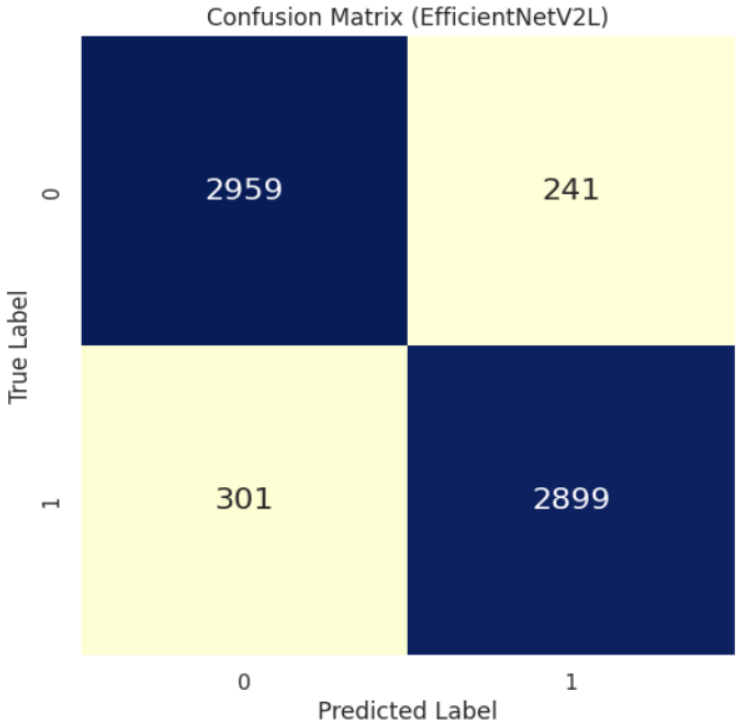


Figure 10: Confusion Matrix – EfficientNetV2L + DNN



The three models were individually evaluated in similar test environments. Results were carefully documented and examined in an effort to obtain a thorough insight into the underlying strengths and weaknesses of each design approach. Interestingly, EfficientNetV2L + DNN attained the highest test accuracy and F1-score overall, followed by EfficientNetB4 + DNN in a distant second position, with DenseNet201 + DNN registering a mid-performance.

6 Results

This chapter documents findings of deepfake detection models used in this study according to quantitative and visual analysis. The results are presented based on the performance of those models in training, validation, and test phases and comparison of the three CNN-based models in this study. The key performance measures and exemplary visualizations are supplied to facilitate easy appreciation of each model's effectiveness and generalizability.

6.1 Model Performance Evaluation

Here we present a comparison between the three deepfake detection models proposed in this work: EfficientNetB4 + DNN, DenseNet201 + DNN, and EfficientNetV2L + DNN. The models are compared based on traditional classification measures of accuracy, precision, recall, and F1-score on the test dataset of over 10,000 images with an equal number of authentic and forged samples.

The results showed that EfficientNetV2L + DNN model achieved highest overall accuracy (about 91.8%) and F1-score with superior generalizability performance and feature extraction capability. The EfficientNetB4 + DNN model was next with very similar performances, while DenseNet201 + DNN model, although performing well in general, exhibited signs of overfitting and relatively low precision and recall.

Table 3 demonstrates comparative performance of three deepfake detection models relying on feature extraction with EfficientNetB4, DenseNet201, and EfficientNetV2L models along with Dense Neural Networks (DNN) for classification.

Table 3: strengths and weaknesses of each model across different evaluation metrics.

Model	Accuracy	Precision	Recall	F1-Score
EfficientNetB4 + DNN	0.91	0.91	0.91	0.91
DenseNet201 + DNN	0.50	0.25	0.50	0.33
EfficientNetV2L + DNN	0.92	0.92	0.92	0.92

To improve the understanding of classification performance, extensive classification reports were generated for all models. The information contains measurements such as precision, recall, and F1-score for "REAL" and "FAKE" class labels in addition to overall accuracy. The reports give precise insight concerning each model's capability to differentiate between original and changed images.

Figure 11: Classification Report – EfficientNetB4 + DNN

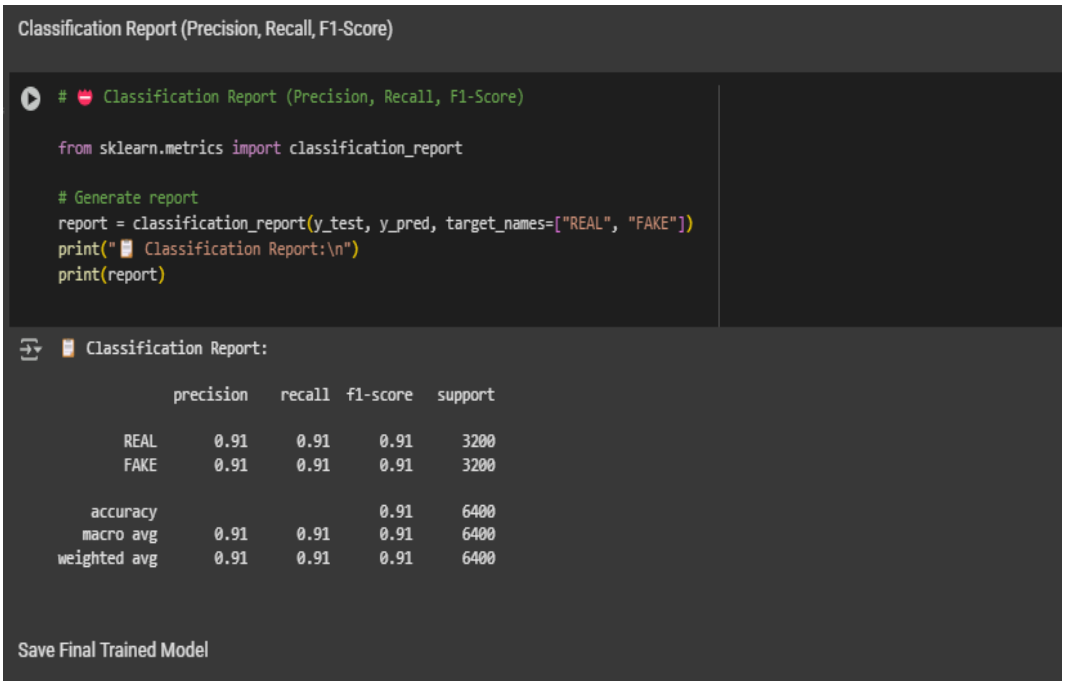


Figure 12: Classification Report – DenseNet201 + DNN

Classification Report (Precision, Recall, F1-Score)

[14]:
```
# 💋 Classification Report (Precision, Recall, F1-Score)

from sklearn.metrics import classification_report

# Generate report
report = classification_report(y_test, y_pred, target_names=["REAL", "FAKE"])
print("📄 Classification Report:\n")
print(report)
```

📄 Classification Report:

	precision	recall	f1-score	support
REAL	0.50	1.00	0.67	3200
FAKE	0.00	0.00	0.00	3200
accuracy			0.50	6400
macro avg	0.25	0.50	0.33	6400
weighted avg	0.25	0.50	0.33	6400

Figure 13: Classification Report – EfficientNetV2L + DNN

Classification Report (Precision, Recall, F1-Score)

[11]:
```
# 📝 Generate and Display Classification Report for EfficientNetV2L

from sklearn.metrics import classification_report

# Generate classification report
report = classification_report(y_test, y_pred, target_names=["REAL", "FAKE"])

# Display the report
print("\n📄 Classification Report (EfficientNetV2L Model):\n")
print(report)
```

📄 Classification Report (EfficientNetV2L Model):

	precision	recall	f1-score	support
REAL	0.91	0.92	0.92	3200
FAKE	0.92	0.91	0.91	3200
accuracy			0.92	6400
macro avg	0.92	0.92	0.92	6400
weighted avg	0.92	0.92	0.92	6400

6.2 Discussion on Results

Experiment outcomes revealed considerable differences in test accuracy, precision, and F1-score among the deepfake detection models evaluated. The EfficientNetV2L-DNN model gave the highest test accuracy, precision, and F1 score, showing a strengthened capacity to differentiate real and generated inputs. Its stable training outcomes and overall validation scores demonstrate a strong generalization capability and a low chance of overfitting even when working with an extended and complicated dataset.

EfficientNetB4 integrated with DNN was confirmed to be impressive in performance, with outcomes merely slightly below EfficientNetV2L scores, yet also with a similarly smooth learning curve. The model also substantiated EfficientNet-based architectures' effectiveness in extracting relevant face features for deepfake classification.

Conversely, the DenseNet201 + DNN model, while achieving high training accuracy, showed poor performance on the test data set. Validation loss showed an increasing trend during the training session, pointing toward possible overfitting and poor generalization ability. This can be attributed to the complexity of the DenseNet architecture that may require more extensive regularization strategies or a different training strategy to work well in this specific scenario.

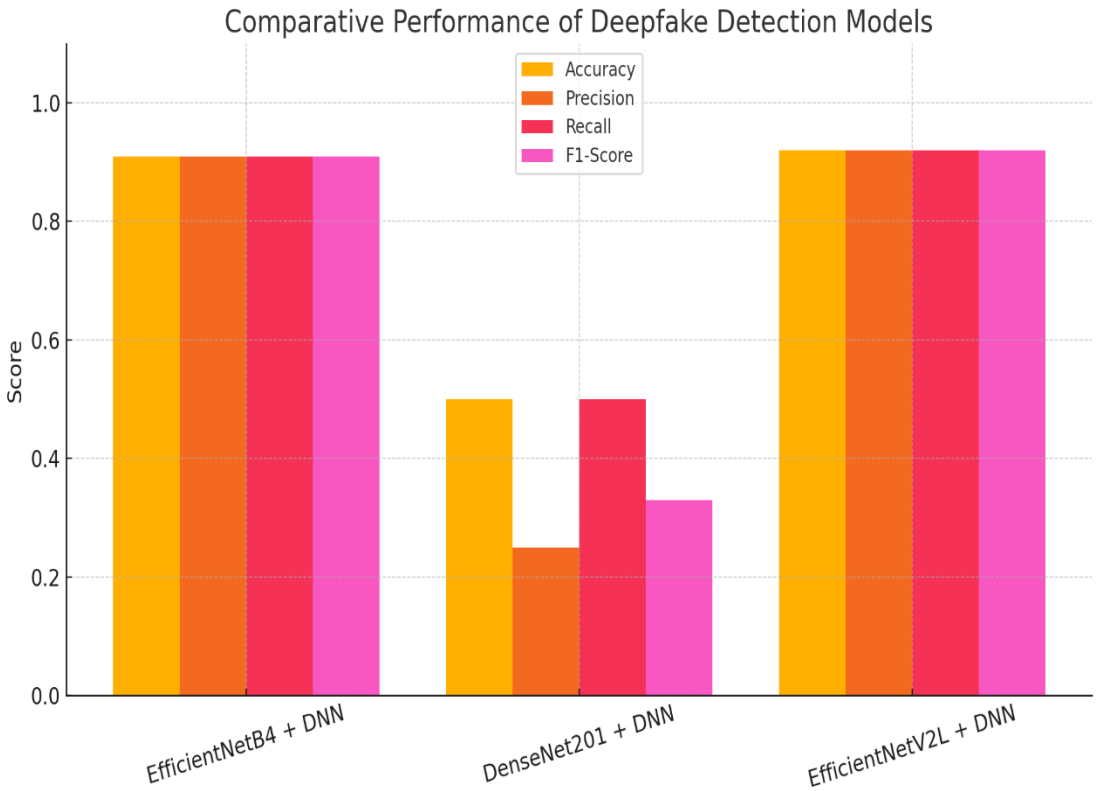
Table 4: Qualitative Comparison of Models Based on Observed Behavior

Model	Accuracy	Overfitting	Stability	Generalization
EfficientNetV2L	✅ High	❌ No	✅ Yes	✅ Strong
EfficientNetB4	✅ Good	❌ No	✅ Yes	✅ Good
DenseNet201	❌ Poor	✅ Yes	❌ No	❌ Weak

The comparison reveals that utilization of lighter and more optimized CNN backbones such as EfficientNetB4 and V2L with a simple dense classification head yields the most effective and viable deepfake detection results. The results support selection of EfficientNetV2L + DNN as the most reliable among models considered to be used in real-world scenarios.

This bar chart depicts relative performance of the three tested models on core performance measures, confirming EfficientNetV2L + DNN to be the top-performing model.

Figure 14: Comparative Performance of Deepfake Detection Models Across Evaluation Metrics



To supplement the comparative evaluation, the raw outputs of each model on the test set are presented below, showing the final test accuracy and loss obtained during evaluation.

Figure 15: Test Evaluation Output – EfficientNetB4 + DNN

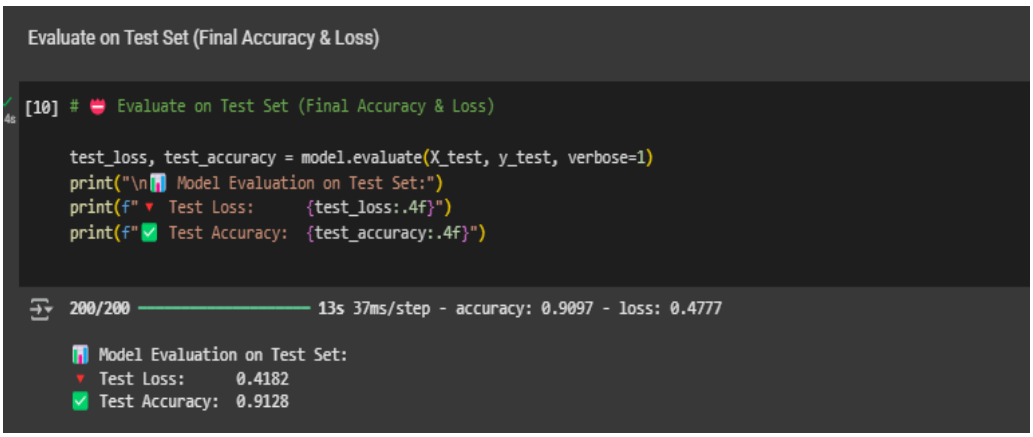


Figure16: Test Evaluation Output – DenseNet201 + DNN

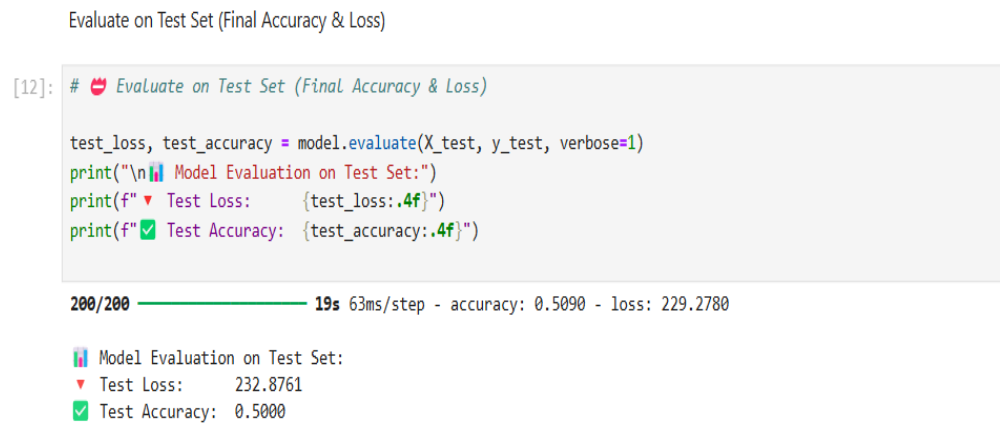
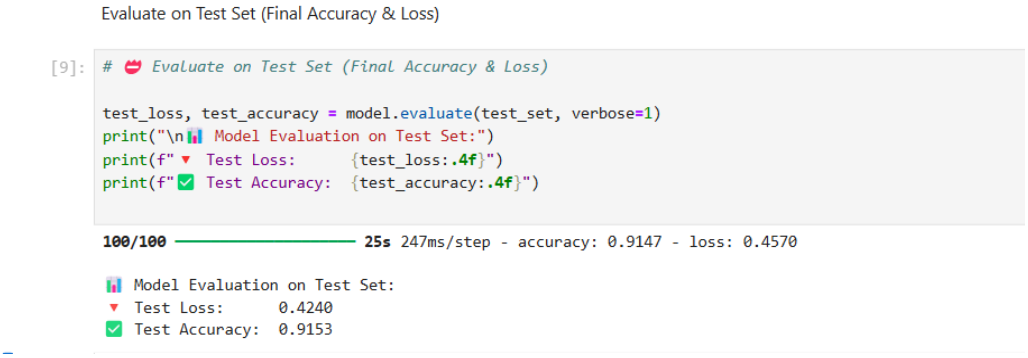


Figure 17: Test Evaluation Output – EfficientNetV2L + DNN



6.3 Respond to Research Questions

How can deepfake detection models be optimized to improve accuracy while maintaining real-time performance?

The work demonstrated that combining EfficientNetV2L, EfficientNetB4, with specifically designed dense neural networks yields high accuracy of detection with acceptable computational efficiency. The use of pretraining through transfer learning drastically

reduced training time as well as increased generalization abilities. Amongst the models analyzed, the combination of EfficientNetV2L with dense networks gave the highest accuracy, making it suitable for use in real-time or near-real-time.

What are the most effective computer vision techniques for identifying manipulated media in cybersecurity applications?

Feature extraction using pretrained CNNs has been shown to be a very effective approach in the field of computer vision. EfficientNet-based models successfully captured subtle visual inconsistencies, such as differences in lighting and unusual facial textures, that often indicate tampering or alterations. These features, when used as input to DNN classifiers, yielded predictions with a high level of accuracy.

How can AI-driven deepfake detection be integrated into existing cybersecurity frameworks to enhance protection?

cybersecurity systems enhance security protocols? The current work outlines an architecture developed for integration with existing cybersecurity measures, in particular with regard to identity verification and media authentication operations. The system enhances the effectiveness of existing cybersecurity systems in recognizing and mitigating risks associated with manipulated digital content before its potential to have an unfavorable impact, through input pre-processing, visual feature extraction, and content classification techniques.

7 Summary

The thesis delved into the development and implementation of a deep-fake detection model through cutting-edge computer vision and deep learning algorithms. Synthetic media is increasingly used for manipulative purposes, thus prompting necessary measures to design effective detection systems. The study looked forward to availing itself of the use of pretrained convolutional neural networks EfficientNetB4, DenseNet201, and EfficientNetV2L utilized as feature extractors in a dedicated Dense Neural Network (DNN).

The experimentation involved a balanced real and manipulated face image dataset that was preprocessed and data augmented to improve generalization. The models were also trained in uniform conditions through model checkpointing and early stopping for a

comparison of performance on equal terms. Google Colab Pro and the Puhti supercomputer were utilized as training platforms to facilitate effective configuration of resources for a range of model complexities.

The comparison was in terms of accuracy rate, precision, recall, and F1-score values and was supplemented by confusion matrices and reports. EfficientNetV2L + DNN was superior in all of the most critical measures followed by EfficientNetB4 + DNN. DenseNet201 + DNN yielded a very good training accuracy rate but was hampered by overfitting and lower generalizability.

This work demonstrates productive utilization of compact dense classifiers in conjunction with lean and optimized CNN backends to obtain high accuracy in deepfake detection. The outcome suggests potential utilization of such models in real-world cybersecurity applications like identity proofing and fraud prevention platforms. There is scope for future work in model extension to video stream processing, examining explainability techniques, and utilization of multimodal data (audio-visual data, etc.) to improve resiliency.

References

Afchar, D., Nozick, V., Yamagishi, J., & Echizen, I. (2018). MesoNet: A Compact Facial Video Forgery Detection Network. *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, 1–7. https://doi.org/10.1109/WIFS.2018.8630761

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. *2017 International Conference on Engineering and Technology (ICET)*, 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Comput. Surv.*, *41*(3), 15:1-15:58. https://doi.org/10.1145/1541880.1541882

Chesney, R., & Citron, D. (2018, December 11). Deepfakes and the New Disinformation War. *Foreign Affairs*, *98*(1). https://www.foreignaffairs.com/articles/world/2018-12-11/deepfakes-and-new-disinformation-war

Fattahi, J. (2024). *Machine Learning and Deep Learning Techniques used in Cybersecurity and Digital Forensics: A Review* (No. arXiv:2501.03250). arXiv. https://doi.org/10.48550/arXiv.2501.03250

Goodfellow, Bengio, & Courville. (2016). *Deep Learning*. https://www.deeplearningbook.org/

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). *Generative Adversarial Networks* (No. arXiv:1406.2661). arXiv. https://doi.org/10.48550/arXiv.1406.2661

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, *313*(5786), 504–507. https://doi.org/10.1126/science.1127647

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications* (No. arXiv:1704.04861). arXiv. https://doi.org/10.48550/arXiv.1704.04861

Karras, T., Laine, S., & Aila, T. (2019). *A Style-Based Generator Architecture for Generative Adversarial Networks* (No. arXiv:1812.04948). arXiv. https://doi.org/10.48550/arXiv.1812.04948

Kietzmann, J., Lee, L. W., McCarthy, I. P., & Kietzmann, T. C. (2020). Deepfakes: Trick or treat? *Business Horizons*, *63*(2), 135–146. https://doi.org/10.1016/j.bushor.2019.11.006

Kim, G., Lee, S., & Kim, S. (2014). A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. *Expert Systems with Applications*, *41*(4, Part 2), 1690–1700. https://doi.org/10.1016/j.eswa.2013.08.066

Kingma, D. P., & Welling, M. (2022). *Auto-Encoding Variational Bayes* (No. arXiv:1312.6114). arXiv. https://doi.org/10.48550/arXiv.1312.6114

LeCun, Y., Bengio, Y., & Hinton, G. (2015a). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

LeCun, Y., Bengio, Y., & Hinton, G. (2015b). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Mohsin, M. (2023). *Deep learning based real-time defect detection, classification, and segmentation methods for industrial quality control*. Itä-Suomen yliopisto. https://erepo.uef.fi/handle/123456789/30531

Nguyen, Nguyen, Q. V. H., Nguyen, D. T., Nguyen, D. T., Huynh-The, T., Nahavandi, S., Nguyen, T. T., Pham, Q.-V., & Nguyen, C. M. (2022). Deep Learning for Deepfakes Creation and Detection: A Survey. *Computer Vision and Image Understanding*, *223*, 103525. https://doi.org/10.1016/j.cviu.2022.103525

Nguyen, Yamagishi, J., & Echizen, I. (2019). Capsule-forensics: Using Capsule Networks to Detect Forged Images and Videos. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2307–2311. https://doi.org/10.1109/ICASSP.2019.8682602

Nirkin, Y., Keller, Y., & Hassner, T. (2022). *FSGANv2: Improved Subject Agnostic Face Swapping and Reenactment* (No. arXiv:2202.12972). arXiv. https://doi.org/10.48550/arXiv.2202.12972

Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, *22*(10), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

Rawat, W., & Wang, Z. (2017). Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation*, *29*(9), 2352–2449. https://doi.org/10.1162/neco_a_00990

Rössler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., & Nießner, M. (2019). *FaceForensics++: Learning to Detect Manipulated Facial Images* (No. arXiv:1901.08971). arXiv. https://doi.org/10.48550/arXiv.1901.08971

Saito, T., & Rehmsmeier, M. (2015). The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLOS ONE*, *10*(3), e0118432. https://doi.org/10.1371/journal.pone.0118432

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4510–4520. https://doi.org/10.1109/CVPR.2018.00474

Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, *6*(1), 60. https://doi.org/10.1186/s40537-019-0197-0

Szeliski, R. (2022). *Computer Vision: Algorithms and Applications*. Springer International Publishing. https://doi.org/10.1007/978-3-030-34372-9

Tan, M., & Le, Q. V. (2020). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks* (No. arXiv:1905.11946). arXiv. https://doi.org/10.48550/arXiv.1905.11946

Tolosana, R., Vera-Rodriguez, R., Fierrez, J., Morales, A., & Ortega-Garcia, J. (2020). *DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection* (No. arXiv:2001.00179). arXiv. https://doi.org/10.48550/arXiv.2001.00179

Verdoliva, L. (2020). Media Forensics and DeepFakes: An Overview. *IEEE Journal of Selected Topics in Signal Processing*, *14*(5), 910–932. https://doi.org/10.1109/JSTSP.2020.3002101

Westerlund, M. (2019). The Emergence of Deepfake Technology: A Review. *Technology Innovation Management Review*, *9*(11), 40–53. https://doi.org/10.22215/timreview/1282

Xu, Y., Deng, B., Wang, J., Jing, Y., Pan, J., & He, S. (2022). *High-resolution Face Swapping via Latent Semantics Disentanglement* (No. arXiv:2203.15958). arXiv. https://doi.org/10.48550/arXiv.2203.15958

Zhou, P., Han, X., Morariu, V. I., & Davis, L. S. (2017). Two-Stream Neural Networks for Tampered Face Detection. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 1831–1839. https://doi.org/10.1109/CVPRW.2017.229

Appendices

Appendix 1: Data Management Plan

The current project employed an openly available dataset of deepfake images, which was obtained from online repositories, such as websites like Kaggle and other academic research materials. The dataset was safely stored in a project folder on local storage, with further backups maintained on Google Drive. The images were all preprocessed and labeled into two different classes: “REAL” and “FAKE.” Note that the dataset included no personal data or identifiable information.

The datasets ready for preprocessing were saved in the.npy format for quick loading and to improve training efficiency. The project followed ethical standards by only using open-source datasets that are meant for academic use. There was no human annotation or user data collection involved.

Appendix 2. Dataset Loading and Preprocessing Code (TensorFlow)

The provided Python code outlines steps to load the data, apply preprocessing steps, and transform it into TensorFlow tf.data.Dataset format. This approach helped in effective utilization of memory and easy integration into training pipeline. The .npy files contained preprocessed deepfakes labeled as "REAL" or "FAKE."

```python
import numpy as np
import tensorflow as tf

# Load preprocessed datasets
x_train = np.load("x_train.npy")
y_train = np.load("y_train.npy")
x_val = np.load("x_val.npy")
y_val = np.load("y_val.npy")
x_test = np.load("x_test.npy")
y_test = np.load("y_test.npy")

print("Datasets loaded successfully!")

# Define generator function
def generator(x, y):
    for i in range(len(x)):
        yield x[i], y[i]

# Dataset settings
batch_size = 4
preprocess = tf.keras.applications.efficientnet.preprocess_input

# Training dataset
train_set = tf.data.Dataset.from_generator(
    lambda: generator(x_train, y_train),
    output_signature=(
        tf.TensorSpec(shape=(224, 224, 3), dtype=tf.uint8),
        tf.TensorSpec(shape=(), dtype=tf.int64)
    )
).map(lambda x, y: (preprocess(tf.cast(x, tf.float32)), y))\
 .shuffle(1000, seed=42).batch(batch_size).prefetch(1)

# Validation dataset
val_set = tf.data.Dataset.from_generator(
    lambda: generator(x_val, y_val),
    output_signature=(
        tf.TensorSpec(shape=(224, 224, 3), dtype=tf.uint8),
        tf.TensorSpec(shape=(), dtype=tf.int64)
    )
).map(lambda x, y: (preprocess(tf.cast(x, tf.float32)), y))\
 .batch(batch_size)

# Test dataset
test_set = tf.data.Dataset.from_generator(
    lambda: generator(x_test, y_test),
    output_signature=(
        tf.TensorSpec(shape=(224, 224, 3), dtype=tf.uint8),
        tf.TensorSpec(shape=(), dtype=tf.int64)
    )
).map(lambda x, y: (preprocess(tf.cast(x, tf.float32)), y))\
 .batch(batch_size)

print("Datasets ready for training!")
```

Appendix 3. Model Training and Checkpointing Code

The following code shows how to train a model with TensorFlow/Keras through the .fit() function. The code includes a ModelCheckpoint callback that systematically keeps the best

model based on validation accuracy. The training routine was run for a total of 47 epochs with both training and validation data.

```python
import os
from tensorflow.keras.callbacks import ModelCheckpoint, Callback

save_dir = r"E:\deepfake_detection_project\saved_models"
os.makedirs(save_dir, exist_ok=True)


class CustomCheckpoint(Callback):
    def on_epoch_end(self, epoch, logs=None):
        if (epoch + 1) % 3 == 0 or (epoch + 1) == 47:
            acc = logs.get("val_accuracy")
            if acc is not None:
                save_path = os.path.join(save_dir, f"model_epoch{epoch+1:02d}_valacc{acc:.2f}.h5")
                self.model.save(save_path)
                print(f"\nModel saved at epoch {epoch+1}: {save_path}")

# Callback to save only the best model
best_model_checkpoint = ModelCheckpoint(
    filepath=os.path.join(save_dir, "best_model.h5"),
    monitor="val_accuracy",
    save_best_only=True,
    verbose=1
)

# Train the model with the callback
history = model.fit(
    train_set,
    validation_data=val_set,
    epochs=47,
    callbacks=[CustomCheckpoint(), best_model_checkpoint]
)
```

Appendix 4. Performance Evaluation Code (EfficientNetV2L)

The following Python code visualizes the training and validation accuracy/loss of the EfficientNetV2L + DNN model over 47 epochs. It provides insights into model convergence, overfitting, and generalization using line plots.

```python
# 👄 Evaluate Model Performance (Accuracy & Loss)

import seaborn as sns
import matplotlib.pyplot as plt

# Set style
sns.set(style="whitegrid")
plt.figure(figsize=(12, 6))

# 📈 Plot Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

# 📉 Plot Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')

plt.tight_layout()
plt.show()

# Final Accuracy Display
final_train_acc = history.history['accuracy'][-1]
final_val_acc = history.history['val_accuracy'][-1]
print(f"✅ Final Training Accuracy: {final_train_acc:.4f}")
print(f"✅ Final Validation Accuracy: {final_val_acc:.4f}")
```

Appendix 5. Confusion Matrix Visualization Code (EfficientNetV2L)

The following code allows visualization and generation of the confusion matrix of the EfficientNetV2L model in combination with the DNN model. The predicted labels are compared with the ground truth values obtained from test data, and it shows them using Seaborn with a varied palette.

```python
# 👄 Confusion Matrix Visualization

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Predict on test set
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Generate confusion matrix
conf_mat = confusion_matrix(y_test, y_pred)

# Plotting
plt.figure(figsize=(6, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', cbar=False, annot_kws={"size": 16})
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```