BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION Computer Science, English

DIPLOMA THESIS

Deepfake Detection in Facial Images

Supervisor
Lect. Dr. CIUCIU Ioana

Author
Rares-Dan-Tiago Goia

2025

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA Informatica in Limba Engleza

LUCRARE DE LICENŢĂ

Detecția de Deepfake-uri în Imagini cu Fețe Umane

Conducător științific
Lect. Dr. CIUCIU Ioana

Absolvent
Rares-Dan-Tiago Goia

2025

ABSTRACT

Recent advancements in artificial intelligence have allowed the creation of highly realistic synthetic images, known as deepfakes, which bring significant challenges to digital trust and security. This thesis addresses the problem of deepfake detection in human facial images by developing and evaluating deep learning models capable of distinguishing between real photographs and those generated by the best generative architectures, such as Generative Adversarial Networks and Diffusion models. To support rapid experimentation, a diverse dataset was constructed, combining real faces and synthetic images produced by multiple state-of-the-art generators. Multiple neural network architectures, including replicated models from the best recent papers and a novel custom design, were compared for their effectiveness in both binary (real vs. fake) and multiclass (predicting the generator model) classification tasks. The best performing models were integrated into a user-friendly web application, providing easy image verification and interpretable predictions. The results demonstrate high accuracy in deepfake detection and highlight the importance of generalization and explainability. This work contributes with new benchmarks for deepfake detection and a publicly accessible tool for image verification in the age of generative AI.

Contents

Chapter 1

Introduction

1.1 The Rise of Generative Facial Synthesis

In recent years, artificial intelligence has made remarkable progress, particularly in its ability to generate content that looks, sounds, and feels remarkably real. One of the most fascinating and popular result of this progress is the rise of AI-generated images of people who do not exist, yet appear convincingly realistic. Moreover, public figures were affected by this advancement as generated images capture them in situations that are not real and in which they were never involved, potentially causing reputation harm.

1.2 Motivation and Context

This impressive discovery is known as the field of generative AI, which enables machines to create new unique content rather than just analyze existing data as traditional models used to do. As the threat of these fake generated images increased, so has the risk; whether somebody is a victim, or just gets fooled and manipulated by them. The need to detect and prevent this from happening has become essential.

At the core of this digital revolution are neural network-based systems inspired by the way the human brain processes information. Like how humans can learn patterns from experience, neural networks learn from large amounts of data, allowing them to find subtle details and reproduce them. Generative models, a more complex subset of neural networks, are designed to create new examples that mimic the data they were trained on. For example, after being shown millions of real human faces, a generative model can learn to create entirely new, synthetic images that look almost like real ones.

The power of generative models comes from their ability to learn complex statistical distributions out of real-world data. Various techniques such as Generative

Adversarial Networks (GANs), Variational Autoencoders (VAEs) and Stable Diffusion models have represented the key in achieving realistic facial synthesis. These models learn to map random noise to realistic images by iteratively refining their output based on feedback.

While the ability to generate realistic faces opens up exciting creative opportunities in fields like entertainment, art, and design, and holds potential for applications such as generating diverse training datasets for AI, it also presents significant societal impact that needs careful consideration.

1.3 Problem Statement & Research Gap

The aim of this thesis is to detect and mitigate the spread of fake facial images that most of the time get people fooled. As previously mentioned, the ability to generate synthetic media can help in lots of scenarios, like datasets creation or augmentation, film, design, but is also comes with the malicious ability to impersonate or deceive those that encounter these. Although recent research has focused on developing deep learning models for deepfake detection, existing solutions often lack transparency, providing little insight into why a particular image is classified as real or fake. This limitation makes it difficult for users to trust automated decisions and for practitioners to improve models.

To address these challenges, this thesis explores not only the effectiveness of multiple deep learning architectures, including both state-of-the-art and custom-designed models, but also prioritizes explainability. By integrating visual explanation methods such as Grad-CAM, the work aims to provide interpretable feedback that highlights which facial regions influenced each prediction, bridging the gap between black-box AI and trustworthy, actionable deepfake detection.

1.4 Objective & Scope

The main objective of this thesis is to design, develop, and evaluate effective deep learning methods for detecting AI-generated faces, with a particular emphasis on achieving strong generalization across different types of generative models. To achieve this goal, a diverse dataset was assembled, consisting of both real and synthetic facial images, collected from publicly available datasets, provided by the generative models creators, as well as generated specifically for this research.

This work focuses on two core classification tasks. The first task is binary classification, where the aim is to distinguish between real and AI-generated (fake) images. The second task extends the problem to multiclass classification, which involves not

only detecting fake images but also identifying the specific generative architecture, such as various GANs or diffusion models, responsible for producing each synthetic face.

1.5 Contributions

A central contribution of this thesis is the construction of a comprehensive and diverse dataset that combines real facial images with synthetic ones generated by several state-of-the-art architectures, including both GANs and diffusion models.

Another significant contribution is the experimental analysis conducted throughout the project. The study compares multiple deep learning architectures, ranging from widely adopted models such as ResNet and EfficientNet to a custom-designed convolutional network, in order to assess their effectiveness for deepfake detection across different scenarios.

Additionally, the thesis addresses both binary and multiclass classification tasks. Extensive experiments were performed to measure how well each model distinguishes real from fake images, as well as how accurately it can attribute synthetic images to their generative sources, using consistent evaluation metrics.

A further contribution represents the emphasis on explainability. By integrating visual explanation methods such as Grad-CAM, the thesis provides interpretable feedback on model predictions, highlighting which facial regions most influenced each decision. This approach provides greater trust in the automated results and enhances the transparency of the detection process.

Finally, the research extends beyond theoretical investigation by delivering a user-friendly web application that allows individuals to upload facial images and receive real-time detection results. This tool demonstrates the real-world applicability of the proposed methods and bridges the gap between research and practice.

1.6 Thesis Structure

- **Chapter 2:** Provides the theoretical foundations necessary for understanding deepfake detection, including an overview of convolutional neural networks (CNNs), fundamental deep learning concepts, and the architecture of modern generative models such as GANs and diffusion models.
- **Chapter 3:** Reviews the current state of research in deepfake detection, summarizing the latest advancements in the field, evaluating the strengths and limitations of existing approaches, and identifying open challenges that motivate the present work.
- **Chapter 4:** Details the methodology and experimental setup, including the construction and preprocessing of the dataset, the implementation of various deep learning models, and the design of binary and multiclass classification experiments. The chapter further analyzes the results obtained from each experimental scenario.
- **Chapter 5:** Describes the development and deployment of the full-stack web application that puts the deepfake detection models in action. This includes system architecture, database and security considerations, frontend and backend implementation, and integration with cloud services for real-world usability.
- **Chapter 6:** Presents a comprehensive summary of the main findings, discusses the implications of the results, and outlines potential directions for future research and practical improvements in the field of deepfake detection.

Chapter 2

Theoretical Implications

2.1 Convolution Neural Networks

Convolutional Neural Networks (CNNs) are a type of neural network designed to handle data that has a grid-like structure, such as images. They use layers of small filters that slide across the data, making them very effective for image recognition, classification, and analysis.

A Convolutional Neural Network (CNN) is composed of several key components, each playing their role in the network's ability to extract relevant features from images. At the core of a CNN are convolutional layers, which scan the input data with learnable filters to automatically detect essential features such as edges, corners, and textures. These raw features are then transformed by activation functions. Nonlinear operations like ReLU allow the network to capture complex, nonlinear patterns in the data. As information flows deeper into the network, pooling layers come into play, consistently reducing the spatial dimensions of feature maps. This process extracts the most important information making the model less sensitive to small variations or distortions in the input images. Finally, the extracted features are fed into one or more fully connected layers, which integrate and interpret the information from all previous layers, allowing the CNN to make accurate and confident final predictions. By working together, these components empower CNNs to excel at challenging visual recognition tasks.

CNNs are powerful for visual tasks because they automatically learn to extract the most important features from images at different levels, starting from basic edges and moving up to complex shapes or objects, without needing hand-crafted rules.

2.2 Fundamental Concepts

2.2.1 Artificial Neurons and Layers

The basic unit of any neural network is the artificial neuron. Each neuron takes input values, multiplies them by weights, adds them up, and passes the result through an activation function. Layers of neurons are stacked to form deep networks, which can capture relationships that are more and more complex.

Neural networks, especially deep ones, build up their understanding in stages. Early layers usually capture simple features, like lines or color blobs, and deeper layers combine these into more abstract concepts.

2.2.2 Activation Functions

Activation functions such as ReLU, sigmoid, or tanh introduce non-linearity. This is an important step because, without non-linearity, no matter how many layers you add, the network would behave just like a linear model and could not capture complex relationships. The choice of activation function can affect both how quickly a network learns and how well it represents complicated patterns. In practice, ReLU is the most used in modern deep networks due to its simplicity and effectiveness, while other functions may be chosen for specific tasks or to solve issues like vanishing gradients.

2.2.3 Regularization and Overfitting

When a network memorizes the training data instead of learning general patterns, it's called overfitting. Regularization methods help prevent this. Common examples are dropout (randomly dropping out neurons during training), batch normalization (normalizing intermediate outputs), and data augmentation (generating new samples from existing data). These strategies encourage the model to focus on essential, generalizable features rather than noise or rare details. Careful use of regularization is especially important when working with limited data or highly complex models, as it directly influences the reliability of predictions on new, unseen inputs.

2.2.4 Evaluation Metrics

Some typical metrics for evaluating classification models are accuracy, precision, recall, F1 score, and ROC-AUC. These metrics help quantify how well the model is performing on both the training and unseen data, making it easier to compare models and understand specific strengths and weaknesses. Table 2.2.4 summarizes the main evaluation metrics used throughout this thesis, including their definitions.

Metric	**Definition**
Accuracy	$\frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$
Precision	$\frac{\text{TP}}{\text{TP}+\text{FP}}$
Recall (Sensitivity)	$\frac{\text{TP}}{\text{TP}+\text{FN}}$
F1-Score	$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision}+\text{Recall}}$
ROC-AUC	Area under the Receiver Operating Characteristic (ROC) curve
Confusion Matrix	2×2 table showing the counts of TP, FP, TN, FN

Table 2.1: Evaluation metrics used in classification tasks

Each of these evaluation metrics provides different insights into model performance. Accuracy gives an overall measure of how many predictions were correct, but can be misleading if the classes are imbalanced. Precision focuses on the proportion of true positive predictions among all positive predictions, making it important when false positives are costly. Recall, also called sensitivity, measures the ability of the model to correctly identify all true positives. The F1 score balances precision and recall, providing a single metric that is especially useful when there is an uneven class distribution or when both false positives and false negatives are important to consider. ROC-AUC evaluates the trade-off between true positive and false positive rates across different thresholds, giving a global view of model's ability to differentiate between classes. Finally, the confusion matrix visually presents all correct and incorrect classifications, serving as the basis for calculating most of the other metrics and making it easier to analyze model errors in detail.

2.3 Training a Neural Network

In supervised learning, a model is trained on a labeled dataset that consists of input-output pairs, where each input has a known correct answer. The objective is for the neural network to learn a mapping from inputs to outputs that generalizes well to new, unseen data rather than simply memorizing the training examples.

The loss function measures how far the model's predictions are from the correct answers. For classification tasks, cross-entropy loss is most common, penalizing incorrect or uncertain predictions.

Optimizers like Stochastic Gradient Descent (SGD) and Adam are used to update the model's parameters to minimize the loss. They work by calculating the gradient of the loss with respect to each parameter and adjusting the parameters in the direction that reduces the loss.

Some values involved in training, known as hyperparameters, are not learned by the model, but must be set manually. The learning rate determines how much the model's parameters are updated during each step, a rate that is too high can cause instability, and one that is too low can cause slow convergence. The batch size specifies the number of training samples used in a single update; smaller batch sizes can help the model generalize but may introduce more noise, while larger batches provide more stable updates but require more memory. The number of epochs refers to how many times the entire dataset is passed through the network during training; too few epochs may result in underfitting, while too many can lead to overfitting if the model starts to memorize the data. Careful selection and tuning of these hyperparameters is essential for achieving optimal model performance.

Training techniques are strategies designed to improve a model's generalization, stability, and efficiency during learning. Among the most used methods are data augmentation, early stopping, batch normalization, and dropout.

Data augmentation involves generating new training samples by applying random transformations, such as flipping, rotating, cropping, or applying filters to the original images. This approach increases dataset diversity and helps the model become less sensitive to variations in the data that it may encounter in real-world scenarios.

Early stopping is a strategy where training is stopped as soon as the model performance in a separate validation set stops improving. By monitoring validation loss or accuracy, early stopping helps prevent overfitting and ensures the model retains its ability to generalize to new, unseen data.

Batch normalization is a technique that normalizes the outputs of each layer to have a consistent mean and variance during training. This accelerates learning, allows higher learning rates, and makes training more stable by keeping the values in each layer more consistent.

Dropout refers to randomly "dropping" a part of the neurons in a layer during each training step. This means the model can't rely too much on a single path between layers and instead learns to find patterns in different ways. As a result, the network becomes better at handling new data and less likely to overfit.

2.4 Generative Architectures

Generative models are deep learning models that can create new data samples that look similar to a given dataset. Instead of just classifying data, they learn the underlying distribution and generate new examples that resemble the original data.

2.4.1 Generative Adversarial Networks (GANs)

Generative Adversarial Networks, or GANs, are a class of neural networks designed to create highly realistic synthetic data. A GAN is made up of two competing neural networks: a generator and a discriminator. The generator's job is to take in random noise and try to produce fake data that looks as similar to the real data as possible. Meanwhile, the discriminator acts as a critic, evaluating how similar the new generated data are to the original data.

During training, these two networks play a kind of game against each other. The generator constantly tries to "fool" the discriminator with more convincing fake images, while the discriminator gets better at spotting the fakes. This adversarial process pushes the generator to improve the realism of its outputs step by step.

GANs perform especially well at generating realistic images because the adversarial feedback encourages the generator to capture even the subtle details and complex patterns found in real data. Over many rounds of training, the generator learns to model the true distribution of the data, allowing it to create new samples that are nearly indistinguishable from real ones to both the discriminator and, often, to human observers as well.

2.4.2 Diffusion Models

Diffusion models are a more modern category of generative models that have quickly gained popularity for their ability to produce exceptionally high-quality and detailed images. The core idea behind diffusion models is to start with pure random noise and then gradually convert this noise into a meaningful image through a step-by-step process. This is done using a neural network that learns how to "denoise" the image at each stage.

The process works in two phases: a forward process and a reverse process. During the forward process, random noise is slowly added to real images over many steps until the images are completely unrecognizable. The reverse process, which the model learns, is the key to generation: starting from random noise, the network learns to remove the noise bit by bit, gradually reconstructing a realistic image at each step.

Diffusion models perform well because this gradual denoising allows the model to capture fine details and complex structures that are often missed by other generative techniques. The stepwise approach makes training more stable and lets the model refine its predictions at each stage. As a result, diffusion models are able to generate images that are both highly realistic and diverse, rivaling or even surpassing the quality of GANs in many applications.

2.4.3 Other Generative Approaches

Other important generative models include autoencoders and variational autoencoders (VAEs). Autoencoders work by learning to compress input data into a lower-dimensional representation and then reconstruct the original data from this compressed form. Variational autoencoders build on this idea by introducing randomness into the encoding process, allowing the model not just to reconstruct inputs, but also to generate entirely new and diverse samples by sampling from the learned latent space. This probabilistic approach makes VAEs particularly useful for generating new data that still follows the patterns found in the original dataset.

Chapter 3

Related Work

The rise of deep generative models like Generative Adversarial Networks (GANs) and diffusion models has changed image synthesis completely [MKAK22, PGM+24, ZYW+24]. Today, it's possible to create hyper-realistic fake images that can be almost impossible to tell apart from real ones, both for people and for traditional computer algorithms [BJK23, DLS+20]. Because of this, detecting AI-generated facial images is now a really important research problem, with real impact on cybersecurity, social trust, and digital forensics [MKAK22, PGM+24].

3.1 Defining Deepfakes and their Threats

Deepfakes come in many forms, including full face synthesis, face swapping, attribute editing, and changing expressions [MKAK22, LLWT15]. Modern methods use tools like StyleGAN, ProGAN, and StarGAN for GANs, as well as Latent Diffusion and Stable Diffusion for diffusion models, to create faces that look more realistic than ever [ZYW+24, PGM+24]. Recent benchmarks and surveys, like GenFace, have shown that deepfakes are now considered a major AI threat, with real-world risks ranging from spreading misinformation to identity theft and targeted scams [ZYW+24, BJK23, PGM+24, MKAK22].

Surprisingly, not all humans are good at spotting deepfakes. In one study, people could only tell apart StyleGAN2-generated faces from real ones about 62 percent of the time, and their confidence in their answers didn't match their actual performance [BJK23]. This shows just how important it is to have reliable, automated deepfake detectors [BJK23, MKAK22].

3.2 Analysis of Generative Model Artifacts

A popular direction in deepfake detection is to look for unique "artifacts" left behind by generative models. For example, Guarnera et al. (2020) presented a method that finds subtle fingerprints in images, left by the way GANs process data using convolutions [GGB20]. Their approach extracts local features using an Expectation Maximization algorithm and can tell not only if an image is fake, but also which specific architecture produced it, including StyleGAN, StyleGAN2, StarGAN, ATTGAN, and GDWCT [GGB20]. Their method is explainable and works well even on images of different sizes and in real-world, uncontrolled conditions, much like classical camera forensics [GGB20].

Other research has found that generative models can also be detected in the frequency domain. Using methods like Fast Fourier Transform (FFT) and Discrete Cosine Transform (DCT), it's possible to spot strange patterns in the spatial frequency and noise statistics of fake images. These frequency-based approaches help detectors generalize to new types of deepfakes and manipulations [PGB25, ZYW+24].

3.3 Data-Driven Deepfake Detection

Most modern deepfake detection now uses deep learning [MKAK22, DLS+20, ZBD24]. Early approaches focused on Convolutional Neural Networks (CNNs) to spot small differences in textures and structures, using architectures like XceptionNet, EfficientNet, and ResNet. These models perform best when trained on big and varied datasets, handling both binary (real vs fake) and multiclass (which generator made it) tasks [DLS+20, PGM+24, ZYW+24].

Lately, Vision Transformers (ViTs) have been used for deepfake detection as well. As shown by Arshed et al. (2024), ViTs are good at modeling global relationships and can focus their attention anywhere in the image, which helps in picking up on both subtle and obvious fake patterns. In multiclass detection, ViTs outperformed CNNs, reaching nearly perfect F1 scores of 99.9 percent, thanks to their self-attention mechanism, which is good at finding inconsistencies across the image that CNNs can sometimes miss [AMI+24].

For real-world use, efficiency is also a big deal. Lanzino et al. (2024) tackled this by using Binary Neural Networks (BNNs), which keep high accuracy while using much less computing power (as much as 20 times fewer operations). They combined these with FFT and Local Binary Pattern (LBP) features for fast, real-time detection on mobile devices, with little loss in accuracy [LFD+24].

3.4 The Challenge of Diffusion Models

One of the hardest problems now is making detectors that work on all kinds of generative models. Early detectors focused mostly on GANs, but diffusion models like Stable Diffusion and DALL-E 2 bring new challenges. These newer models don't leave the same obvious patterns (like checkerboards or grid artifacts) that GANs do, and detectors trained just on GANs often fail on diffusion-based fakes [PGB25, ZYW+24, PGM+24]. Robust detection now means handling a whole mix of generative methods and adapting quickly to new types [PGB25, ZYW+24, PGM+24].

To address this, the GenFace benchmark built a large dataset covering both GAN and diffusion fakes, with detailed labels for each manipulation and generator type. Their results show that there is still a lot of work to do, especially when it comes to building features and models that generalize well across both different generators and datasets [ZYW+24].

3.5 Towards Real-World Deployment

Porcile et al. (2024) looked at how to actually deploy deepfake detectors on social media platforms. Their model, based on EfficientNet-B1 and trained on a mix of real and fake images from both GANs and diffusion models, was able to stay accurate and robust even when images were compressed, resized, or attacked with laundering techniques. The big message from their work is that detectors need to balance accuracy, speed, scalability, and interpretability if they are going to work in real, high-traffic online environments [PGM+24].

Mundra et al. (2023) also worked on explainability, using low-dimensional embeddings to capture regularities in GAN-generated faces (like alignment and facial feature positions). This made their system not only accurate, but also more transparent; helpful for platform transparency and user trust [MPM+23].

3.6 Summary and Research Gap

In summary, the literature points to a few main trends:

- Artifact-based forensic analysis is still really important for explaining and attributing deepfakes [GGB20, PGB25].

- Deep learning models, from classic CNNs to transformers and efficient BNNs, now achieve state-of-the-art results but need careful design to generalize and to work at scale [PGM+24, AMI+24, LFD+24, DLS+20, ZBD24].

- Diffusion models create new problems for detection and motivate new benchmarks, like GenFace, focused on real-world data [ZYW+24, PGB25].

- People are not reliable deepfake detectors, which makes automated systems essential [BJK23, MKAK22].

This thesis builds on all of these advances by: (1) systematically comparing how well real and AI-generated faces can be detected across several generative models, (2) benchmarking both CNNs and transformer-based architectures, and (3) focusing on making models that generalize to new generators and work well in real-world scenarios. The end goal is to contribute to building practical, trustworthy, and explainable deepfake detection systems.

Chapter 4

Experiments and Individual Contributions

4.1 Overview and Experimental Setup

This chapter presents the experimental framework and main contributions of this work on deepfake detection, covering dataset construction, training methodology, model evaluation, and comparison with state-of-the-art methods. Experiments are grouped into two main tracks:

- **Binary Classification:** Distinguishing real from fake facial images.
- **Multiclass Classification:** Identifying the generative source of each image.

All experiments use a modular, reproducible training pipeline with practical design for real-world use, implemented using the PyTorch library.

4.2 Dataset

4.2.1 Data Sources

The dataset merges real and synthetic faces from:

- CelebA: All real face images.
- StyleGAN1, StyleGAN2: Official GAN-generated faces.
- ThisPersonDoesNotExist (TPDNE): GAN-based faces, extracted from Kaggle.
- Stable Diffusion XL (SDXL): Custom diffusion-generated faces.

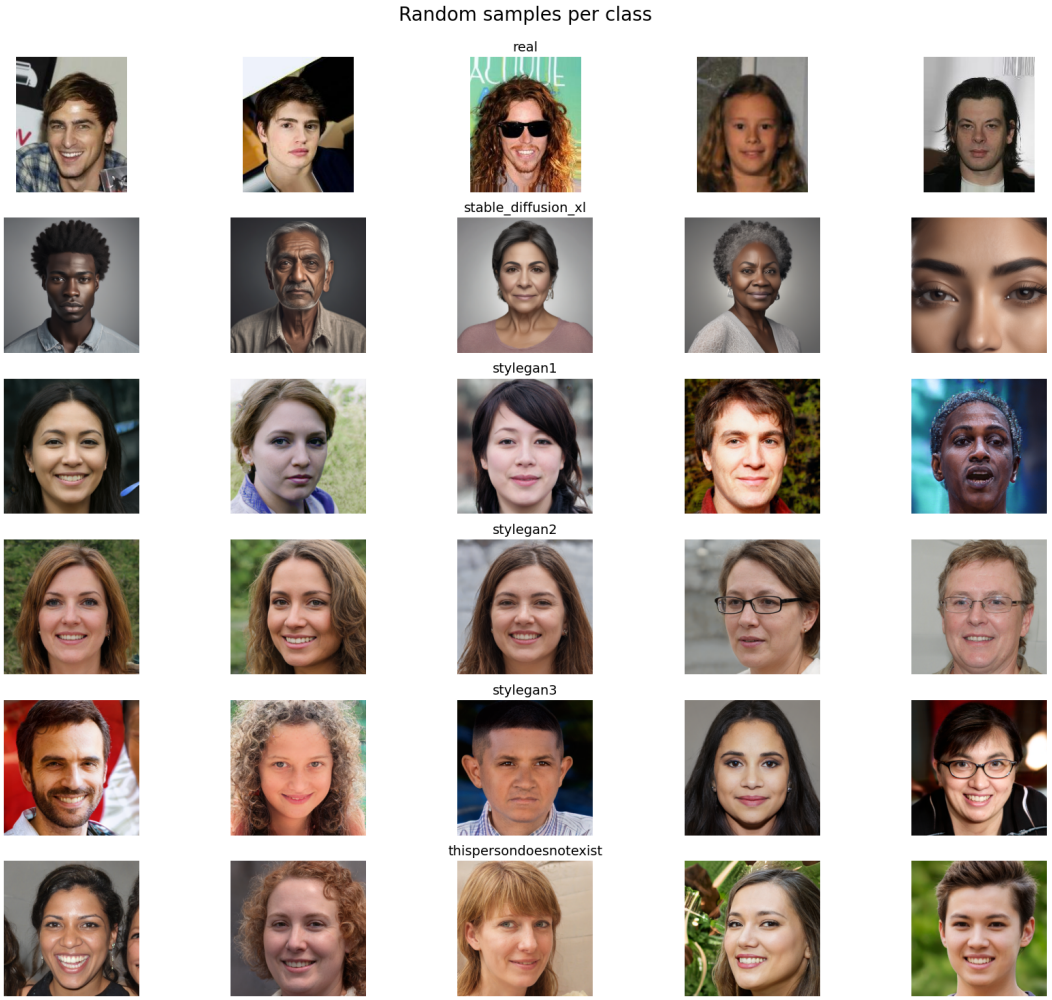

Figure 4.1: Example images from each class in the dataset.

4.2.2 Dataset Sizes and Class Distribution

To reproduce practical scenarios, three dataset sizes were constructed as shown in Table 4.1.

Table 4.1: Dataset composition for each experiment track.

	Real	**StyleGAN1**	**StyleGAN2**	**TPDNE**	**SDXL**
Small	500	100	100	100	100
Medium	1500	300	300	300	300
Large	5000	1000	1000	1000	1000

4.2.3 Data Allocation and Preprocessing

For all experiments, the data was split into 70% training, 15% validation, and 15% test subsets, ensuring that no image appeared in more than one subset. This separation prevents any potential data leakage and ensures that all evaluation metrics reflect genuine model generalization to unseen data.

Before splitting, all images were uniformly resized to 218×178 pixels which represents the dimensions of the smallest images in the constructed dataset. This resizing guarantees a consistent input size for all neural network models and avoids shape-related errors across multiple architectures. Each image was then converted to a tensor for compatibility with PyTorch-based models.

The training set (70%) was used to fit model parameters, the validation set (15%) for hyperparameter tuning, model selection, and early stopping, and the test set (15%) strictly for the final, unbiased evaluation of model performance. Dataset splitting was performed randomly but deterministically, using a fixed random seed to ensure reproducibility.

All dataset splits, resizing, and experiment results were logged and versioned using MLFlow. This approach ensures that every result is traceable to a specific dataset partition and preprocessing pipeline, reinforcing transparency, reproducibility, and scientific rigor throughout the research.

4.3 Experiment Tracking with MLFlow

To ensure reproducibility and effective management of experiments, MLFlow was utilized. This open-source platform was designed for the complete machine learning life cycle. MLFlow allows logging of hyperparameters, metrics, model checkpoints, code versions, and even data splits. Each experiment run is recorded and can be compared visually through the MLFlow UI, making it easy to analyze the effects of different configurations and model choices. The tool also supports storing and versioning models, which is essential for reliably selecting the best-performing solutions for deployment and integration. In the context of this thesis, MLFlow was valuable for tracking results across different dataset sizes, architectures, and training strategies, providing full transparency and reproducibility throughout the entire research process. Figure 4.2 illustrates the MLFlow tracking interface used during these experiments.

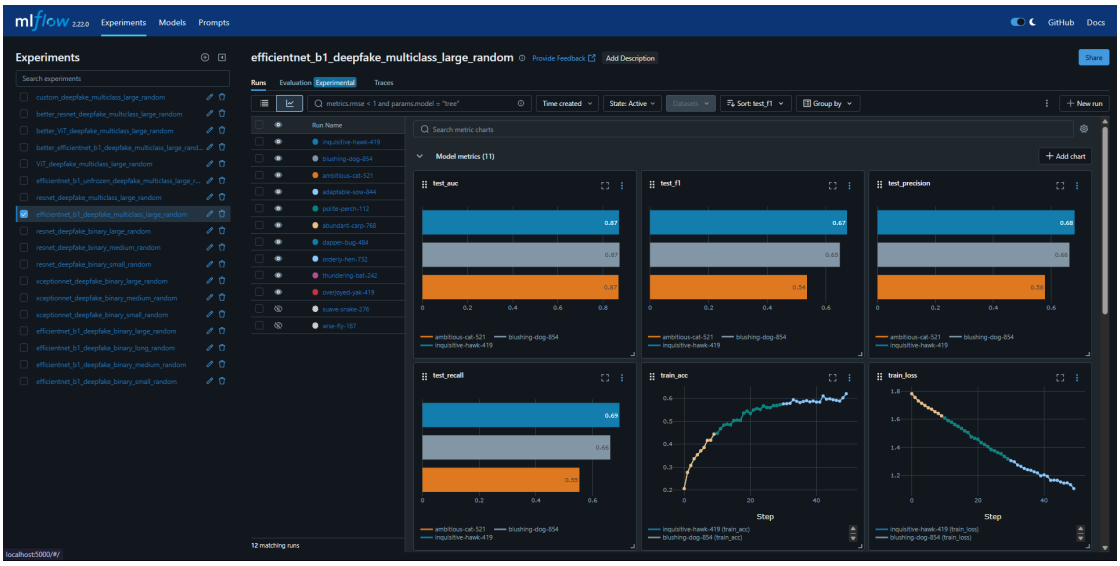

Figure 4.2: Example MLFlow tracking UI showing logged experiment metrics and comparisons.

4.4 Training and Evaluation Pipeline

4.4.1 Pipeline Structure and Workflow

The entire training and evaluation process was implemented in a modular PyTorch-based framework. Data loading was handled with custom Dataset classes, supporting flexible class sampling and splitting.

4.4.2 Training Loop

For each experiment, the training loop began with model initialization, where the neural network architecture was configured according to the requirements of either binary or multiclass classification. Data was loaded in shuffled batches to ensure diverse and effective parameter updates throughout training. Each batch of images was passed through the model in a forward pass to generate predictions. The loss was then computed using Binary Cross Entropy for binary classification tasks or Cross Entropy Loss for multiclass problems. Moreover, gradients were calculated through backpropagation, and model parameters were updated using the selected optimizer, with learning rate scheduling applied to facilitate efficient convergence. To ensure the best performance, model weights and optimizer states were checkpointed whenever validation loss improved, and early stopping was employed to prevent overfitting. All relevant hyperparameters, checkpoints, and performance metrics were systematically logged using MLFlow, providing full transparency and enabling comparison between experimental runs.

4.4.3 Evaluation Metrics

Upon completion of training, each model was evaluated on the test set. Performance metrics included:

- Accuracy
- ROC-AUC
- Recall
- Precision
- F1-score
- Confusion matrix

Metrics were computed on the entire validation and test splits to allow easy performance analysis.

4.5 Binary Classification Experiments

4.5.1 Experimental Setup

Binary classification was performed on the small and medium datasets, as summarized in Table 4.1, with the goal of distinguishing real faces from all fake sources combined. Three different neural network architectures were evaluated for this task: XceptionNet, ResNet-50, and EfficientNet-B1. Each model was trained and assessed using the same experimental setup to ensure fair comparison of their ability to differentiate between real and synthetic facial images.

4.5.2 Results and Discussion

Model	**Validation F1 Score**
EfficientNet-B1	97%
ResNet-50	95%
XceptionNet	88%

Table 4.2: Validation accuracies for binary classification after 15 epochs on the medium dataset.

EfficientNet-B1 achieved the best performance, followed by ResNet-50 and XceptionNet. All models obtain great results, in line with the findings of Porcile et al. [PGM+24] for high-quality, balanced datasets.

4.6 Multiclass Classification Experiments

Inspired by the setup described in the Porcile et al. [PGM+24] paper, I extended the scope of the experiments to multiclass classification. The aim was not only to detect fake images but also to identify which specific generative model from StyleGAN1, StyleGAN2, ThisPersonDoesNotExist, or Stable Diffusion XL produced each sample. For this task, I retrained both the EfficientNet and ResNet architectures, modifying the final linear layer of each model to output logits for every class, rather than a binary real-or-fake prediction. Cross-entropy loss was used in place of binary cross-entropy to adapt to the multiclass nature of the problem.

The results demonstrated that both EfficientNet and ResNet were capable of distinguishing between real and various types of fake images, although there was increased confusion among some of the generative classes, particularly the different StyleGAN variants.

4.7 Vision Transformer Experiments

To further explore model generalization, I trained a Vision Transformer (ViT) [AMI+24] for the multiclass deepfake detection task. Unlike traditional convolutional neural networks, ViTs operate on image patches using self-attention mechanisms, which enables them to capture complex dependencies across the image. For this experiment, the ViT model was initialized with weights pretrained on ImageNet and then finetuned on my multiclass dataset using the same training and evaluation procedures as for the CNN architectures. The ViT achieved strong results and, consistent with findings in recent literature, performed particularly well in distinguishing between fakes generated by different models.

4.8 Custom CNN Architecture Experiments

Wanting to explore solutions beyond established backbone models, I designed and implemented a custom convolutional neural network referred to as DeepfakeNet. The architectural concept for DeepfakeNet was motivated by the archiecture of most CNNs-based models, which involves stacking multiple convolutional and pooling layers to progressively extract increasingly abstract features from input images. Additionally, this architecture was inspired by the use of attention mechanisms in Vision Transformers (ViT), and includes an attention branch that helps the network focus on the most important areas of the image for detecting deepfakes.

A notable design decision was the use of a four-channel input, with the standard three RGB channels augmented by an additional channel containing the Fast

Fourier Transform (FFT) representation of each image [PGB25]. This choice was based on findings in the literature that frequency-domain information is effective for highlighting subtle artifacts typical of synthetic face generation. The core of DeepfakeNet consists of three convolutional blocks, each integrating batch normalization, ReLU activation, and Squeeze-and-Excitation (SE) modules [HSS18] to enable channel-wise recalibration and improve feature discrimination. Inspired by attention heads in transformer-based models, a parallel convolutional branch generates an attention map that further guides the network's focus during learning. After feature extraction, the representations are flattened and processed through a two-layer fully connected classifier to output the final class probabilities.

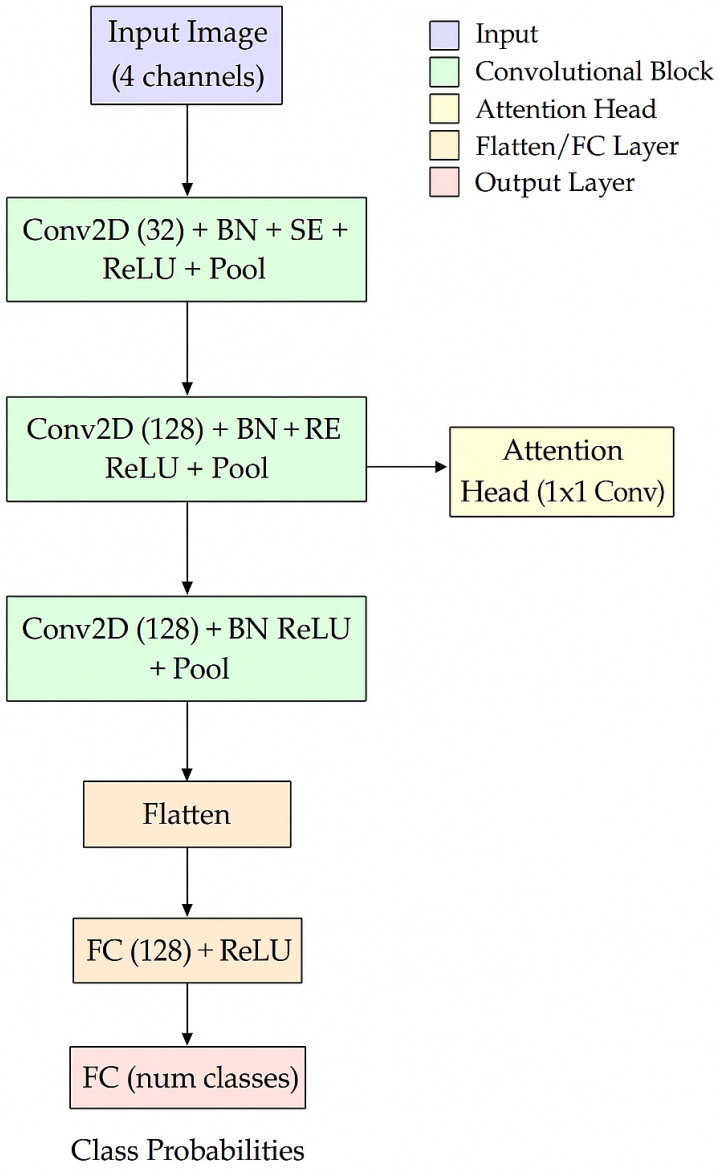

Figure 4.3: Schematic diagram of the custom DeepfakeNet architecture.

DeepfakeNet was trained on the multiclass dataset using the same training and evaluation pipeline as the other convolutional baselines, ensuring a fair comparison. While the custom architecture did not outperform pretrained models such as EfficientNet and ResNet in terms of overall accuracy, it provided valuable insights into how attention mechanisms and frequency-based features can be leveraged in deepfake detection. The experiment highlighted both the potential and the current limitations of custom CNN designs in this application domain.

4.8.1 Comparison with State-of-the-Art Methods

The main aspect of this thesis is the evaluation of the developed models against state-of-the-art deepfake detection methods reported in recent literature. Table 4.8.1 summarizes multiclass classification accuracy and F1-scores for the models implemented in this work, compared with representative results from prior studies.

Model / Study	**Accuracy (%)**	**F1-score (%)**
EfficientNet-B1 (Mine)	87	93
ResNet-50 (Mine)	86	88
ViT (Mine)	91	92
DeepfakeNet (Mine, custom)	80	82
Porcile et al. [PGM$^+$24] (EfficientNet-B1)	88.4	–
Arshed et al. [AMI$^+$24] (ViT)	99.9	99.9

Table 4.3: Comparison with State-of-the-Art Deepfake Detection Methods (Multiclass Classification)

The results indicate that the models developed in this thesis achieve competitive multiclass metrics compared to leading approaches from recent literature. It is important to note, however, that direct comparison between studies should take into account differences in dataset composition and complexity:

- **This thesis:** All models were evaluated on a challenging and diverse dataset composed of real images from CelebA and a variety of synthetic images generated by several state-of-the-art models, including StyleGAN1/2, ThisPersonDoesNotExist, and Stable Diffusion XL. This composition provides a valuable test of both GAN- and diffusion-based deepfake detection.
- **Porcile et al. [PGM$^+$24]:** Their EfficientNet-B1 model was benchmarked using the GenFace dataset, a large-scale benchmark specifically designed to include both GAN- and diffusion-generated forgeries as well as real faces from multiple sources. Their work sets a strong baseline for real-world generalization.

- **Arshed et al. [AMI+24]:** Their Vision Transformer model was trained and tested on an extensive, custom-collected dataset exceeding 10,000 images, containing a broad mix of real and synthetic faces from multiple GAN and diffusion architectures. Their reported F1-score is near perfect, reflecting both the capacity of ViTs and the scale of their dataset.

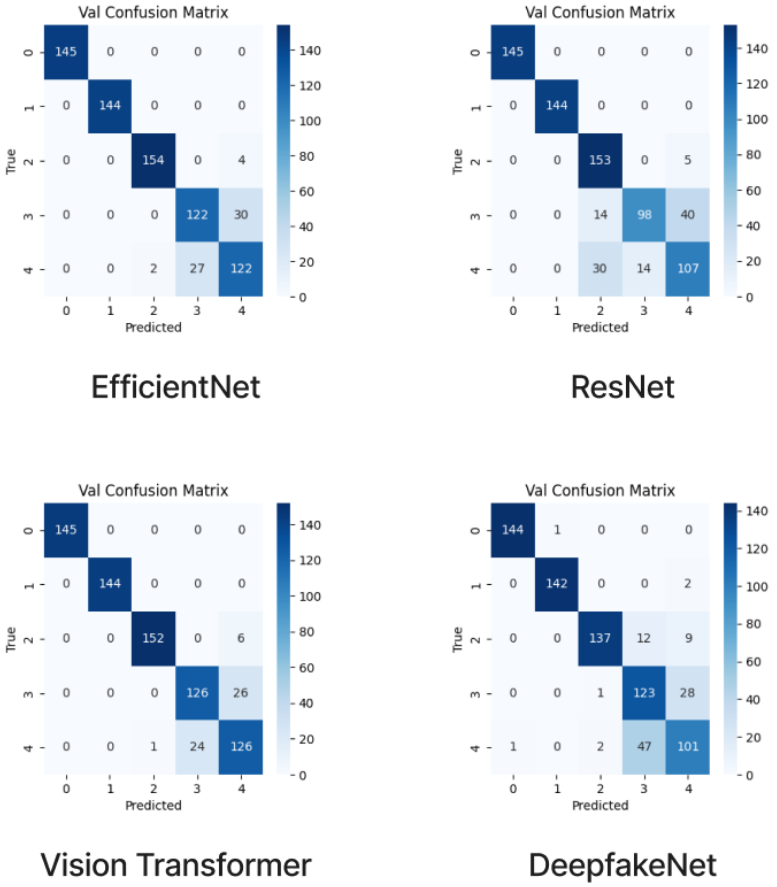

Figure 4.4: Confusion Matrices across tested models.

These differences in dataset sources, size, and the diversity of fake classes significantly impact the reported metrics across studies. Overall, the comparative analysis validates the effectiveness of the proposed models in the fight against AI-generated facial forgeries.

4.9 Summary and Insights

This chapter detailed the entire experimental process done for deepfake detection, from dataset construction and preprocessing to the implementation and evaluation of several leading deep learning architectures. By testing both state-of-the-art

models such as EfficientNet, ResNet, and Vision Transformer, as well as a custom-designed convolutional network, the study explored the strengths and limitations of each approach in both binary and multiclass classification tasks. The experiments were conducted on datasets of varying size and complexity, reflecting real-world scenarios and providing an assessment of model generalization.

Vision Transformer demonstrated the highest accuracy and F1-score on multi-class detection, followed by EfficientNet and ResNet, while the custom DeepfakeNet offered valuable lessons about the effects of architectural design choices and attention mechanisms. The experiments showed that having a diverse, well-collected dataset is important, and that using explainable AI methods helps make the models more transparent and trustworthy.

Overall, this experimental framework creates a strong foundation for practical deepfake detection, highlighting both the current capabilities and remaining challenges in reliably identifying synthetic facial images generated by a range of modern generative models.

Chapter 5

Implementation of the Web Application

5.1 Overview

The Deepfake Detection task was wrapped in a modern full-stack web application, designed to enable users to upload images that contain faces and verify if these are AI-generated or not. Besides the clear verdict (real or fake) a detailed and interpretable result of the analysis is provided, including the top three most probable classes predicted by the model and their corresponding confidence scores, allowing the user to see which generative architecture was most probably used to create the image. Moreover, an optional Grad-CAM heatmap is displayed, showcasing which facial features impacted the models decisions the most.

The application was deployed in the cloud, providing public online access to anyone who wants to test their own facial images. The only perquisite is the creation of a personal account in order to access the system's features, enabling an individualized experience. Users have the option to save their detection results for later reference.

Supabase service was used to provide authentication, database hosting, storage buckets and to keep the application endpoint secure, by protecting all requests and responses.

5.2 System Design

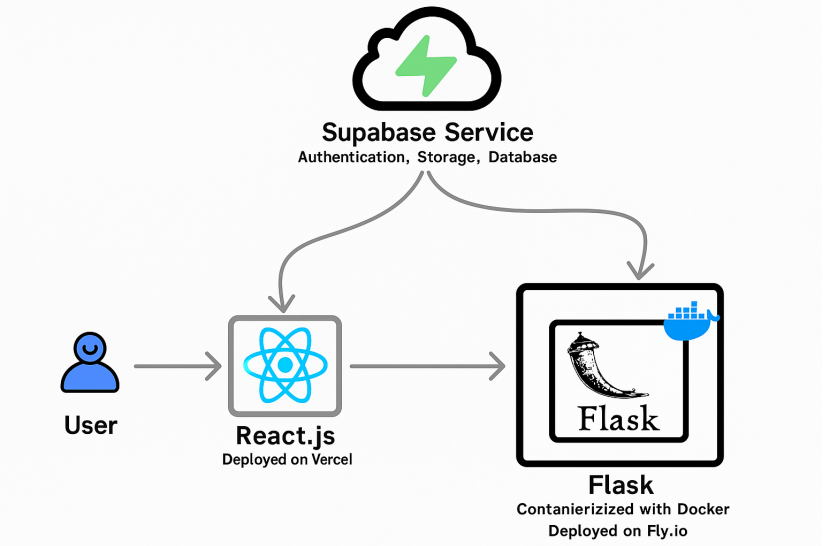

Figure 5.1: High-Level System Design

5.2.1 Database Design

The application uses Supabase's PostgreSQL database for all its data storage operations. The database is structured to make user management and history tracking straightforward and secure.

Supabase provides a pre-built authentication tables that handle user accounts, passwords, and sessions automatically both for OAuth providers and simple email and password authentication. This means Supabase already takes care of storing or securing passwords. Each user is assigned a unique user-id, which is used everywhere else in the application to link actions and data back to the right person.

In addition to the default auth tables, a custom Images table was designed specifically for tracking deepfake detection history. The table contains a foreign key for the user-id which is linked to Supabase's authentication table. Each time a user decides to save an analyzed image and it's metadata, a new record is added to this table. Also, a user can delete a history element, removing the record from the table.

Uploaded images and generated ones in the report are added to a Supabase Storage Bucket, such that there is no need for local storage. The Images table stores the paths to the corresponding images from the storage such that the reports are clearly retrieved and displayed for the user.

By organizing the database this way, every prediction a user makes is safely stored and easily accessible, while also being kept private from other users. The use of unique user IDs guarantees that each user only has access to their own results.

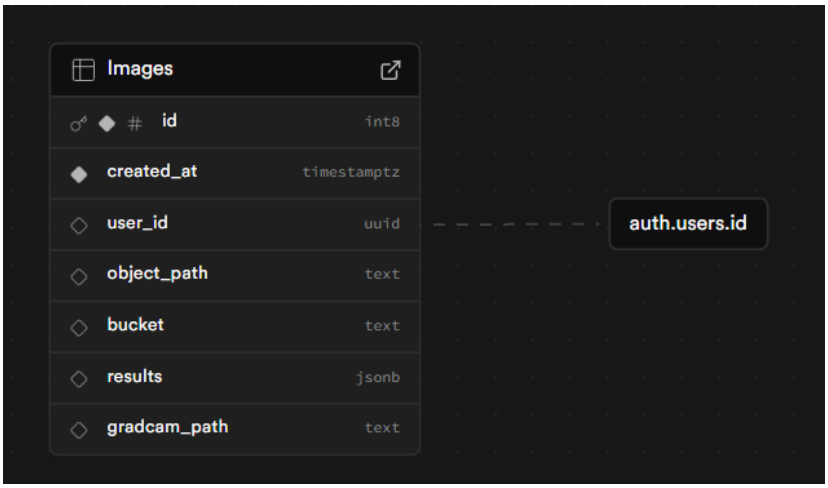

Figure 5.2: Database Structure

This design not only keeps user data isolated and secure, but also makes it easy to scale the application and retrieve a complete history for any user, at any time.

5.2.2 Frontend Design

The frontend was developed using React.js with Vite, providing TypeScript support. The application follows a modular structure, in which every user-interface component has its own dedicated file and its corresponding CSS module, ensuring ease of scalability and maintainability. Supabase service was integrated directly into the fronted as the authentication provider. The React application uses the Supabase client to handle all user login, registration and session management operations. Upon successful authentication, Supabase provides a secure JSON Web Token (JWT), which is then attached to protect the API requests from the backend.

Reusable Components

A set of reusable components were developed in order to provide scalability and modularization for the application:

- **Button**: A customizable button element allowing primary and secondary actions, each representing the style of the displayed button.
- **TopBar**: The navigation bar contains links to Dashboard (the main page), History, and Logout. The logout functionality is provided by the Supabase client, ending the user's active session.
- **UploadArea**: This represents a drag-and-drop or click-to-upload interface for image selection. Allows users to upload only files that have image extensions like: .jpg, .jpeg, .png, etc.

- **ModelSelector**: A drop-down menu allowing the user to select the available models for detection. Each model contains a brief description about itself, such that users understand the advantages and disadvantages of each model.
- **ResultBox**: Displays the resulted report, containing the verdict, best three confidence levels and the Grad-CAM heatmap if selected by the user.
- **HistoryCard**: Shows a summary of the previously saved predictions, just the uploaded image, the verdict and the model used. These are retrieved via REST API calls using the Supabase JWT from a self-defined table.
- **ResultModal**: This is a modal view for detailed prediction and heatmap visualization in the History section. Allows the user to zoom-in or zoom-out the original image and the Grad-CAM heatmap.

Pages and Routing

The main functionalities are organized into distinct pages:

- **Landing Page**: This page contains a brief introduction into the application and call to get started. It also contains links to my GitHub and LinkedIn profiles in the Footer of the page.
- **Authentication (Login/Register)**: Secure sign-in and account creation. It is required before accessing the application features. Registration and login are handled entirely by Supabase Auth, supporting both email/password and OAuth authentication using Google. Confirmation emails are automatically sent and managed by Supabase to newly registered users.
- **Dashboard**: This represents the central hub for image upload, model selection, detection and reporting results. Containing, most important features of the application, this page was structured into more React Hooks to provide a cleaner structure and scalability if more features would be added to it.
- **History Page**: Allows users to review their own, saved, past detections. Each detection shows a summary preview, which, on-click opens a pop-up containing full detailed analysis. User history is linked to their Supabase user ID, ensuring data isolation and privacy.

Access to the Dashboard and History pages is restricted. Only authenticated users are allowed to access their features. This is verified client-side by the presence of a valid Supabase session, and enforced server-side by a valid JWT verification.

React Specifics

- The application contains React Hooks (such as useState, useEffect) for state and side-effect management.
- The React Router (Navigator) is used for clean URL routing and protected routes.
- Each page and component is fully responsive, supporting both desktop and mobile devices.
- Supabase's client library is used for session refresh, automatic token renewal, and secure logout.

5.2.3 Backend Design

The backend was developed using the Flask framework in Python. This was chosen for its simplicity and ease of integration with state-of-the-art machine learning libraries (PyTorch, OpenCV, etc.). The backend was designed with scalability, modularity, and security.

Supabase is a critical dependency also on the backend for secure authentication and data access control. Every request done to the protected endpoints requires a valid Supabase JWT. The backend verifies the JWT, extracts the user ID, and authorizes or denies access accordingly. All user-specific data, such as detection history, is associated with the unique Supabase user ID.

Project Structure

- **Blueprints**: The REST API routes are organized using Flask blueprints, separated by feature category (detect, history).
- **Service Layer**: This is the core business logic. Each feature has its own implemented service module.
- **Routes Layer**: Handles HTTP requests, calls service functions, manages input/output serialization.
- **Models**: ML model classes, including their architectures and loading utilities are included for dynamic selection and inference.
- **Middleware**: Custom authentication decorators validate Supabase JWTs on every protected route, and securely extract the user identity for per-user data operations.

- **Utils**: Functions that don't belong to any service module but are very benefic for the application functionalities.

Authentication and Security

- Only authentication-related endpoints are open; all others require a valid Supabase JWT in order to be accessed.
- Security is enforced via a self-defined wrapper applied to all protected APIs, which checks the session token by verifying the Supabase JWT signature and expiry.
- CORS policies restrict API access to requests from the deployed frontend domain.
- User-specific detection history is stored and queried using the Supabase user ID, ensuring strict data isolation and privacy.

Model Management

Custom and state-of-the-art model architectures are included as importable Python modules, supporting dynamic loading and switching. Model selection is handled via an argument in the detection request. All model weights and architectures are stored locally in the backend for fast inference.

5.3 REST APIs

Detection APIs

- **GET /api/models**: Returns all the models available for the users and a brief description associated to them.
- **POST /api/validate-face**: Receives an image, checks for presence of a human face, returns validation result. Requires Supabase JWT in the Authorization header.
- **POST /api/detect**: Receives an image, validates and preprocesses it, runs model inference, returns the verdict, top three confidences, and optionally the Grad-CAM heatmap if it was selected by the user. Requires Supabase JWT in the Authorization header.

History APIs

- **GET /api/history**: Returns the authenticated user's saved predictions, including image metadata, verdicts, confidences, and any explainability results. User is identified and authorized by their Supabase user ID.
- **POST /api/history/save**: Allows saving a detection result if a user wants to. The data is always stored under the authenticated Supabase user ID.

5.4 Feature Specification

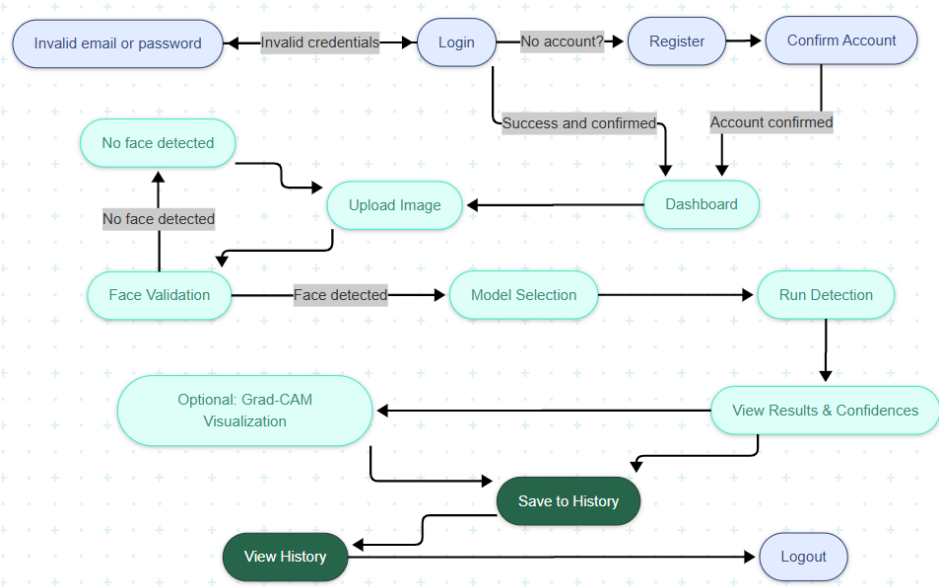

Figure 5.3: Use case diagram of the application

- **Authentication**: Secure sign-up and login using email/password or OAuth (Google), with confirmation e-mail for new accounts. All authentication flows, password management, and email confirmation are implemented via Supabase Auth. JWT tokens are automatically refreshed by the frontend, and are mandatory for all protected API calls.
- **Model Selection**: Users may select from multiple state-of-the-art models (Visual Transformer, ResNet, EfficientNet) or custom architectures via a dropdown menu.
- **Image Validation**: The system verifies that each uploaded image contains a clear, single human face before detection is permitted, minimizing invalid or irrelevant predictions. This is achieved using a YOLOv8-based face detection model available on HuggingFace: arnabdhar/YOLOv8-Face-Detection.

- **Advanced Analysis**: For every detection, the application displays the top three model confidence scores, helping users interpret the prediction. Optionally, users may also request Grad-CAM visualizations to see which facial areas most influenced the model.
- **User History**: Each authenticated user can access a personal history of their saved detections, including images, results, confidence scores, and Grad-CAM heatmaps. User histories are securely isolated in the Supabase-managed database and can only be accessed after validating the user's JWT.

5.5 Application Flow

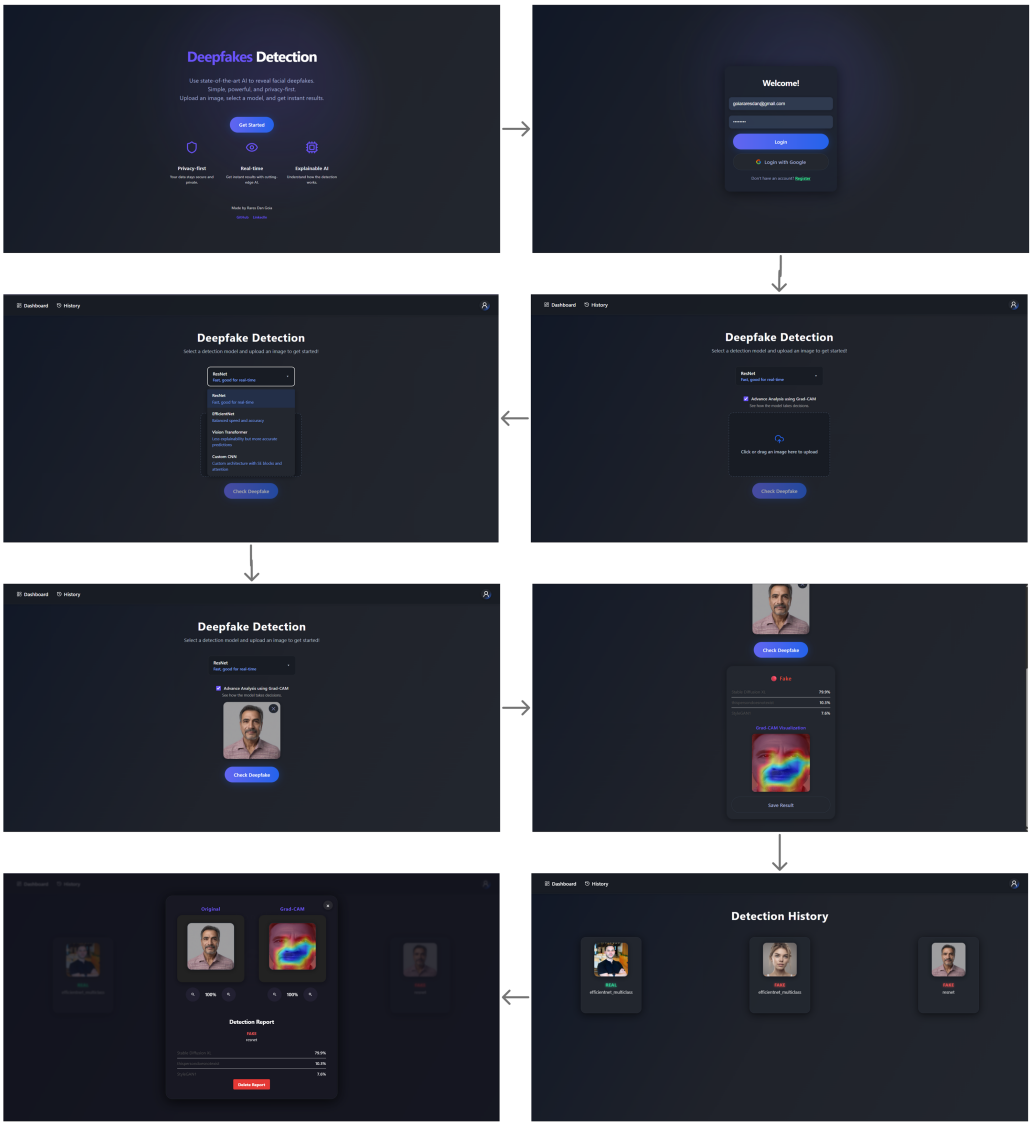

Figure 5.4: Application Flow

5.6 Cloud Deployment and Containerization

5.6.1 Backend Docker Containerization

To ensure consistency and reliability across different environments, the backend is fully containerized using Docker. Docker encapsulates the entire backend application, including all Python modules, dependencies, and environment configurations, into a single, portable image. This approach eliminates issues related to dependency conflicts or mismatched library versions, and allows the backend to be run on any platform that supports Docker. By defining all requirements in a Dockerfile, the application can be built and started with a single command, simplifying updates and scaling.

5.6.2 Backend Deployment

Deployment is handled via Fly.io, chosen for its support of Docker containers, always-on free tier, and low operational overhead. Fly.io assigns a public URL and enables automatic HTTPS. Scaling and updates are achieved via standard Docker workflow and Fly.io CLI. Redeployment is done easily as the project is linked to a GitHub repository and can be triggered automatically or manually when changes are made.

Supabase service credentials, such as the public API key and JWT secret for backend verification, are set via environment variables in the deployment configuration.

5.6.3 Frontend Deployment

The frontend application is deployed using Vercel, a modern cloud platform built for high-performance static and serverless React applications. Vercel provides automatic HTTPS for all deployed projects, integrates with Git to support continuous deployment, and allows for easy configuration of custom domains. Additionally, Vercel makes environment variable management very intuitive, so the Supabase project URL and public API key are securely provided as frontend environment variables and accessed by the Supabase client as needed.

5.7 Summary

This chapter presented the design and implementation of a modern web application for deepfake detection, integrating advanced AI models with an accessible and secure user experience. A full-stack approach was used with React.js and Vercel for the frontend, Flask and Fly.io for the backend, and Supabase for authentication,

data management, and storage. The system ensures scalability, privacy, and modularity. Key features such as model selection, interpretable results with Grad-CAM visualizations, and user-specific prediction history were detailed, highlighting the application's usability and transparency. The deployment process and security considerations were also outlined, demonstrating a production-ready system that supports both research and practical usage in deepfake detection.

Chapter 6

Results & Conclusions

Results

The experiments conducted throughout this thesis demonstrated that deep learning models can effectively distinguish between real and synthetic facial images, even when these are generated by state-of-the-art GAN and diffusion architectures. In binary classification, EfficientNet-B1 achieved the highest validation F1-score of 97%, followed closely by ResNet-50. For the multiclass task, where the objective was to identify the specific generative model used, the Vision Transformer (ViT) outperformed all other models, achieving a classification accuracy of 91% and an F1-score of 92%. This confirms the growing effectiveness of transformer-based architectures in computer vision tasks. Although the custom CNN architecture (DeepfakeNet) did not surpass pretrained models in terms of raw metrics, it provided valuable insights into how frequency-domain representations and attention mechanisms can be used for deepfake detection.

Conclusions

This thesis proposed a complete solution for detecting AI-generated facial images, covering model development, experimental evaluation, and real-world application through a full-stack web platform. By integrating explainable AI techniques such as Grad-CAM and enabling public access via a cloud-deployed system, the project bridged the gap between research and usability. The modular experimental pipeline, experiment tracking via MLFlow, and systematic benchmarking against both CNN and ViT models confirmed the strength and generalization of the best-performing architectures. Furthermore, the use of diverse datasets containing images from GANs and diffusion models ensured that the conclusions are applicable to a wide range of synthetic content generation techniques. Overall, the results support the

thesis objective and demonstrate that deepfake detection models can be both accurate and interpretable.

Future Work

While this thesis focused on image-based detection of deepfakes, several directions for future improvement and research are possible. First, extending the current system to support video-based deepfake detection would be a valuable enhancement, as most real-world misuse occurs in video format. Second, future experiments could explore adversarial scenarios, testing how the models respond to adversarial attacks or image laundering techniques. Moreover, expanding the dataset to include more diffusion architectures and further fine-tuning transformers on larger datasets may improve generalization even more. Finally, incorporating lightweight or mobile-optimized models could make the detection system usable on edge devices, facilitating faster and more scalable real-world adoption.

Bibliography

[AMI+24] Muhammad Asad Arshed, Shahzad Mumtaz, Muhammad Ibrahim, Christine Dewi, Muhammad Tanveer, and Saeed Ahmed. Multiclass ai-generated deepfake face detection using patch-wise deep learning model. *Computers*, 13(1):31, 2024.

[BJK23] Sergi D Bray, Shane D Johnson, and Bennett Kleinberg. Testing human ability to detect 'deepfake'images of human faces. *Journal of Cybersecurity*, 9(1):tyad011, 2023.

[DLS+20] Hao Dang, Feng Liu, Joel Stehouwer, Xiaoming Liu, and Anil K Jain. On the detection of digital face manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern recognition*, pages 5781–5790, 2020.

[GGB20] Luca Guarnera, Oliver Giudice, and Sebastiano Battiato. Deepfake detection by analyzing convolutional traces. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 666–667, 2020.

[HSS18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[LFD+24] Romeo Lanzino, Federico Fontana, Anxhelo Diko, Marco Raoul Marini, and Luigi Cinque. Faster than lies: Real-time deepfake detection using binary neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3771–3780, 2024.

[LLWT15] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738, 2015.

[MKAK22] Asad Malik, Minoru Kuribayashi, Sani M Abdullahi, and Ahmad Neyaz Khan. Deepfake detection for human face images and videos: A survey. *Ieee Access*, 10:18757–18775, 2022.

[MPM+23] Shivansh Mundra, Gonzalo J Aniano Porcile, Smit Marvaniya, James R Verbus, and Hany Farid. Exposing gan-generated profile photos from compact embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 884–892, 2023.

[PGB25] Orazio Pontorno, Luca Guarnera, and Sebastiano Battiato. Deepfeaturex net: Deep features extractors based network for discriminating synthetic from real images. In *International Conference on Pattern Recognition*, pages 177–193. Springer, 2025.

[PGM+24] Gonzalo J Aniano Porcile, Jack Gindi, Shivansh Mundra, James R Verbus, and Hany Farid. Finding ai-generated faces in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4297–4305, 2024.

[ZBD24] Nishigandha N Zanje, Anupkumar M Bongale, and Deepak Dharrao. Detecting facial image forgeries with transfer learning techniques. *Int J Adv Appl Sci*, 13(1):93–105, 2024.

[ZYW+24] Yaning Zhang, Zitong Yu, Tianyi Wang, Xiaobin Huang, Linlin Shen, Zan Gao, and Jianfeng Ren. Genface: A large-scale fine-grained face forgery benchmark and cross appearance-edge learning. *IEEE Transactions on Information Forensics and Security*, 2024.