Deepfake Detection: A Comparative Analysis of Supervised Classification and GAN-based Anomaly Detection

Francesco Simbola
Master's Degree in Artificial Intelligence
Università degli Studi di Verona
francesco.simbola@studenti.univr.it

August 2025

Contents

Abstract

This study investigates the efficacy of two distinct deep learning paradigms for detecting manipulated facial images, analyzing a comprehensive dataset for this task. The primary approach employed a supervised classification framework, where a ResNet-18 convolutional neural network was trained to differentiate between authentic and deepfake images. This method was systematically evaluated across several variants, including transfer learning and attention mechanisms. The second, alternative approach explored a GAN-based anomaly detection methodology, where a discriminator was trained exclusively on authentic images to learn the distribution of real data. The supervised ResNet-18 classifier achieved excellent discriminative capability, yielding a ROC-AUC of 0.9616 and an accuracy of 86.92%. Conversely, the GAN-based anomaly detector failed to generalize, performing at a near-random level with a ROC-AUC of 0.4701. These results strongly suggest that, for deepfake detection, learning explicit discriminative features from a dataset containing both classes is a significantly more robust and effective strategy than unsupervised modeling of only the authentic data distribution.

1 Introduction

The rapid democratization of deep learning models has led to a surge in hyper-realistic synthetic media, colloquially known as **deepfakes**. Their potential for misuse in generating disinformation and sophisticated fraud has made automatic detection a critical area of research, while this technology holds promise for creative applications. The challenge lies not only in the increasing quality of these fakes but also in their diversity, as new generation techniques emerge continuously. [5] This project takes a comparative approach to address a more fundamental question: **what is the most effective learning paradigm for this task?** I investigate **two opposing strategies**. The first is a classic supervised approach, where a **Convolutional Neural Network (CNN)** is trained on a labeled dataset of both authentic and fake images to learn their differentiating features. The second explores a more challenging anomaly detection paradigm, in which a model learns the deep statistical distribution of only authentic images, with the goal of identifying fakes as deviations from that learned norm. This comparison was designed to reveal insights into the nature of deepfake artifacts themselves. My study addresses four specific questions to guide this investigation:

1. How effectively can a standard CNN architecture (ResNet-18) distinguish between real and fake images when trained with explicit labels?

2. What is the performance impact of different training variants, such as using pre-trained weights or adding attention mechanisms?

3. Can a model trained exclusively on real images (the discriminator of a GAN) generalize to detect fakes it has never seen?

4. What do the contrasting results of these two paradigms reveal about the most reliable signals for deepfake detection?

2 Dataset and Preprocessing

The dataset used is "DeepFake and Real Images" from [7]. It consists of 290,335 facial images divided into three standard splits: training, validation, and test sets. Each split contains two classes: "Real" (authentic human faces) and "Fake" (digitally manipulated or synthetically generated faces). [15] The dataset follows a balanced class distribution across all parts, which is particularly important for binary classification tasks where class imbalance can significantly skew model performance:

- Training Set: 140,002 images (70,001 Real + 70,001 Fake)
- Validation Set: 39,428 images (19,787 Real + 19,641 Fake)
- Test Set: 110,905 images (55,524 Real + 55,381 Fake)

Real Fake

Real Fake

Figure 1: Dataset samples comparison

2.0.1 Augmentation Pipeline

I implemented a multi-stage preprocessing pipeline with a dual objective: ensuring proper normalization of the input data while introducing regularization mechanisms to enhance the model's generalization ability. A methodological distinction was established between training and evaluation phases, with the former employing stochastic augmentation strategies and the latter utilizing deterministic procedures to guarantee consistent assessment. [15] The training pipeline was specifically designed to expose the network to a broader range of visual conditions, encouraging the learning of invariant feature representations. The 256×256 pixel images are processed through a RandomCrop to 224×224 pixels (providing spatial augmentation from the larger source image) and a RandomHorizontalFlip with 50% probability to achieve reflection invariance.

```
# For the training set, apply random augmentations to improve model generalization
    if split == "train":
        return T.Compose([
            T.Resize(256),
            T.RandomCrop(size),
            T.RandomHorizontalFlip(p=0.5),
            T.ColorJitter(0.1, 0.1, 0.1, 0.05),
            T.ToTensor(),
            T.Normalize(mean, std),
        ])
```

Listing 1: Training Pipeline

In contrast, the validation and test pipeline eliminates all stochasticity by applying direct resizing to 224×224 pixels followed by deterministic **CenterCrop** operations. This ensures that identical image regions are analyzed across all evaluation runs, providing stable and unbiased measures of the model's true generalization capability while enabling fair comparison of performance metrics across different epochs and experimental variants.

```
    # For validation and test sets, use simple resizing and center cropping for consistent
    evaluation
    else:
        return T.Compose([
            T.Resize(size),
            T.CenterCrop(size),
            T.ToTensor(),
            T.Normalize(mean, std),
        ])
```

Listing 2: Val & Test Pipeline

2.0.2 Advanced Regularization with CutMix

[21] CutMix functions by selecting two images from a batch, cutting a random rectangular patch from one, and pasting it over the corresponding region of the other. The ground truth label for the resulting hybrid image is then recalculated as a weighted combination of the original two labels, with weights proportional to the visible area of each source image in the final composite: $\lambda \times \text{label}_A + (1-\lambda) \times \text{label}_B$, where λ represents the proportion of the original image remaining after patch replacement. This method compels the model to learn from multiple, spatially distinct feature distributions within a single sample and prevents it from becoming overly reliant on specific localized visual artifacts for classification, which is a known failure mode

in deepfake detection. The implementation generates the mixed samples and proportionally adjusted labels on-the-fly during training.

```python
# Mix two images by cutting a patch from one and pasting it over another
def cutmix(x,y,alpha=1.0):
    # Determine mixing proportion from a Beta distribution
    lam = np.random.beta(alpha,alpha)
    rand_index = torch.randperm(x.size(0)).to(x.device)

    # Generate random bounding box coordinates for the patch
    y_a, y_b = y, y[rand_index]
    bbx1,bby1,bbx2,bby2 = rand_bbox(x.size(), lam)

    # Replace the patch in the original images with the patch from shuffled images
    x[:,:,bby1:bby2,bbx1:bbx2] = x[rand_index,:,bby1:bby2,bbx1:bbx2]

    # Adjust the lambda value based on the final patch area
    lam = 1-((bbx2-bbx1)*(bby2-bby1)/(x.size(-1)*x.size(-2)))

    # Return the mixed image and original labels for loss calculation
    return x, y_a, y_b, lam
```

Listing 3: CutMix Implementation

2.0.3 Normalization Strategies

The pixel values, initially in the range [0, 255], are first scaled to [0.0, 1.0] by the ToTensor transformation and then standardized by subtracting the mean and dividing by the standard deviation. The choice of these statistical parameters is not arbitrary but is conditioned on the model's initialization strategy. When employing a ResNet-18 backbone pre-trained on ImageNet, I used the specific mean and standard deviation of the ImageNet dataset: mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225] for RGB channels respectively. This aligns the statistical properties of input data with the distribution on which the model's initial feature extractors were learned, facilitating more effective transfer learning. A symmetric normalization to the range [-1, 1] is employed using mean=[0.5, 0.5, 0.5] and std=[0.5, 0.5, 0.5] for models trained from scratch or for the Generative Adversarial Network. This is a standard practice that centers the data around zero and it is often beneficial for training stability. [1]

3 Approach 1: Supervised DeepFake Classification

Supervised learning enables optimal performance by directly learning from the contrast between positive and negative examples during training. I decided to establish consistent architectural choices, optimization protocols and evaluation metrics across all supervised experiments to ensure valid comparisons between experimental variants. [15]

3.0.1 Architectural Foundation: ResNet-18

The foundational architecture for all supervised variants is **ResNet-18**, a convolutional neural network that has demonstrated good performance. [17] It consists of 18 weight layers with an initial convolutional layer followed by four "blocks" of residual layers. Each block contains multiple BasicBlock units, where each

BasicBlock has of two 3×3 convolutional layers with batch normalization and ReLU activation functions. The residual connection in each BasicBlock allows the input to bypass the two convolutional layers via an identity mapping, enabling the learning of residual functions rather than direct mappings. This design facilitates the training of much deeper networks by ensuring that gradients can flow backward through the network without vanishing. The model was instantiated using the `torchvision.models` library. A critical design decision was the dynamic replacement of the classification head. Rather than hardcoding the final layer dimensions, the implementation programmatically inspects the `in` attribute of the original fully-connected layer to determine the dimensionality of the feature vector produced by the final convolutional block. During the project development, the classification head architecture evolved. The first four models (Variants 1-4) were trained with a simple linear classifier:

```
Simple linear classifier used in Variants 1-4
in_feats = model.fc.in_features
model.fc = nn.Linear(in_feats, 2) # Direct mapping to 2 classes
```

Listing 4: Simple classification head (Variants 1-4)

This 512-dimensional feature vector was directly mapped to a 2-dimensional output space corresponding to "Real" and "Fake" classes. Only the final variant incorporated explicit regularization:

```
Enhanced classifier with dropout regularization for Variant 5
in_feats = model.fc.in_features
model.fc = nn.Sequential(
nn.Dropout(p=0.5), # Explicit regularization
nn.Linear(in_feats, 2) # Final classification layer
)
```

Listing 5: Enhanced classification head (Variant 5 only)

This architectural distinction explains why Variant 5 (CBAM + Dropout) achieved superior performance, as it uniquely combines attention mechanisms with explicit regularization in the classification head. [20] This approach ensures that framework remains adaptable to other ResNet variants or entirely different architectures with minimal code modification, representing a robust and scalable design choice for experimental research.

3.0.2 Optimization and Training Protocol

All supervised models were optimized using the **Adam** algorithm, a first-order gradient-based optimization method that computes adaptive learning rates for each parameter. [8] Adam combines the advantages of two other extensions of stochastic gradient descent: AdaGrad, which works well with sparse gradients, and RMSProp, which works well in non-stationary settings. The algorithm maintains separate adaptive learning rates for each parameter by estimating both the first moment (the mean) and the second moment (the uncentered variance) of the gradients. The objective function for all supervised variants is the **CrossEntropyLoss**, which measures the performance of a classification model whose output is a probability distribution over multiple classes. For binary classification problem, CrossEntropyLoss computes the negative log-likelihood of the correct class, providing stronger gradients for confident misclassifications and encouraging the model to not only classify correctly but to do so with high confidence. A critical component of training protocol is the implementation of a learning rate scheduling strategy. I decided to employ a **StepLR** scheduler, which multiplies the learning rate by a decay factor gamma = 0.1 every 7 epochs. This strategy is based on the empirical observation that reducing the learning rate as training progresses often leads to better convergence

properties. Initially, a higher learning rate allows the model to make large updates and quickly navigate the loss landscape toward promising regions. As training progresses and the model approaches a local minimum, the reduced learning rate enables more precise fine-tuning of the parameters, preventing the optimization from overshooting optimal values.

```
    # Model
    model = get_model(arch=args.arch, pretrained=args.pretrained,
                      use_cbam=args.cbam, num_classes=2).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=args.lr)
    scheduler = StepLR(optimizer, step_size=7, gamma=0.1)
```

Listing 6: Optimizer and learning rate scheduler setup from mycode/train.py

3.1 Experimental Variants

Five distinct model variants were developed and trained. Each variant was designed as a controlled experiment to investigate the impact of specific techniques while holding all other factors constant.

3.1.1 Variant 1: Baseline from Scratch

The underlying hypothesis is that deepfake generation, despite its sophistication, introduces systematic artifacts or statistical patterns that differ from those found in authentic facial imagery. These differences may manifest as subtle inconsistencies in texture, lighting, facial geometry or high-frequency details that a deep neural network can learn to recognize through sufficient exposure to labeled examples. **Detailed Methodology.** This model uses random weight initialization (`pretrained=False`) and is trained for 10 epochs using batch size 64, learning rate 3e-4, with 2,188 batches per epoch processing the complete 140,002-image training set. For this variant, input images are normalized using symmetric scaling to the range [-1, 1]. This is achieved through PyTorch's preprocessing pipeline: `ToTensor()` converts pixels from [0, 255] to [0, 1], then `Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])` applies the transformation (pixel 0.5) / 0.5. This normalization centers data around zero and improves optimization stability. The training follows the common optimization protocol described before, employing only standard augmentation techniques (RandomCrop, RandomHorizontalFlip, ColorJitter) without specialized methods like CutMix or attention mechanisms. This ensures the variant represents a clean baseline as confirmed by the progressive improvement from 88.55% to 98.71% training accuracy and final validation accuracy of 88.20%. The loss trajectory reveals distinct learning phases: rapid improvement during epochs 1-4 (loss: 0.2534 $\rightarrow$ 0.0747), steady optimization in epochs 5-7 (loss: 0.0658 $\rightarrow$ 0.0550), and fine-tuning in the final epochs 8-10 (loss: 0.0381 $\rightarrow$ 0.0321). This pattern aligns with the StepLR scheduler design where the learning rate reduction after epoch 7 enabled more precise parameter optimization.

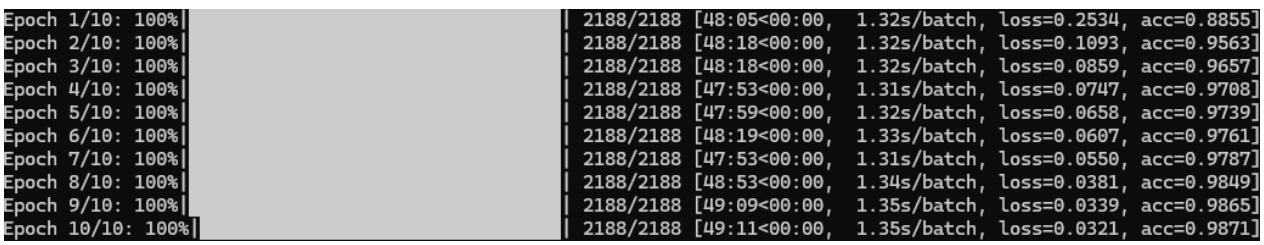


Figure 2: Loss during training for baseline model

3.1.2 Variant 2: Transfer Learning Baseline

The theoretical foundation of transfer learning rests on the observation that deep convolutional networks learn hierarchical feature representations. Early layers typically capture generic, low-level features that are applicable across visual tasks while deeper layers learn increasingly task-specific representations. For deepfake detection, the expectation is that features learned from ImageNet's diverse collection of natural images—particularly provide a superior starting point compared to random initialization. [15] **Detailed Methodology.** This variant employs the same ResNet-18 architecture but initializes the convolutional backbone with weights pre-trained on the ImageNet dataset. ImageNet consists of over 14 million images spanning 1000 object classes, providing exposure to an enormous variety of visual patterns, lighting conditions, and photographic styles. The pre-trained model has already learned to extract rich feature representations through this extensive training process. The transfer learning process involves two distinct phases. In the first phase, the pre-trained ResNet-18 has already learned a comprehensive set of feature extractors through training on ImageNet. These learned parameters represent millions of gradient update steps and capture sophisticated visual patterns that would be computationally expensive to learn from scratch. In the second phase, I perform fine-tuning, where the entire network—including both the pre-trained convolutional layers and the newly initialized classification head—is trained on deepfake dataset. During ImageNet pre-training, images were normalized using dataset-specific statistics: mean=[0.485, 0.456, 0.406] and standard deviation=[0.229, 0.224, 0.225] for the red, green, and blue channels respectively. Deepfake images must be normalized using these same ImageNet statistics rather than the symmetric [-1, 1] normalization used in the scratch baseline.

3.1.3 Variant 3: Attentional Model (CBAM)

Traditional convolutional neural networks process all spatial locations and feature channels uniformly, potentially diluting the signal from critical regions with noise from irrelevant background areas. Attention mechanisms address this limitation by learning to dynamically weight different parts of the feature representation based on their relevance to the task. [14] **Detailed Methodology.** This variant augments the standard ResNet-18 architecture with a **Convolutional Block Attention Module (CBAM)**, a lightweight attention mechanism that sequentially applies channel and spatial attention. CBAM operates on the principle that both "what" (channel attention) and "where" (spatial attention) are important for effective feature selection. The CBAM module is strategically inserted immediately after the final convolutional block (`layer4`) of the ResNet-18, allowing it to operate on high-level semantic features just before the global average pooling operation. The channel attention component generates a channel attention map by leveraging the inter-channel relationship of features. It applies adaptive average pooling along the spatial dimensions to produce a global context descriptor, which is then processed through a shared multi-layer perceptron with a reduction ratio of 16 to generate channel attention weights. The spatial attention component operates differently: it computes both average and max pooling along the channel dimension to create two spatial descriptors, concatenates them, and processes the result through a 7×7 convolutional layer to produce spatial attention weights. The rationale is that different feature channels may have varying importance for detecting manipulation artifacts—some channels might be highly responsive to authentic facial features, while others might be more sensitive to synthetic anomalies—while spatial attention helps focus on the most relevant facial regions.

3.1.4 Variant 4: Advanced Regularization (CutMix)

CutMix addresses a fundamental challenge in deepfake detection. Overfitting's risk to spurious correlations or dataset-specific artifacts rather than learning general principles of facial authenticity. The model is forced to make decisions based on partial information and distributed features rather than relying on any single, localized region. **Detailed Methodology.** CutMix operates by creating new training samples through a process of spatial mixing. For each training batch, pairs of images are randomly selected and a rectangular patch is cut from one image and pasted onto the corresponding region of another image. The size and location of this patch are determined stochastically, with the patch area following a Beta distribution that controls the mixing ratio. The ground truth label for the resulting composite image is calculated as a weighted combination of the original labels, with weights proportional to the visible area of each source image. This label mixing ensures that the training signal remains consistent with the visual content, requiring the model to learn to make fractional predictions that correspond to the actual composition of the image.

```
if args.cutmix:
    inputs, la, lb, lam = cutmix_fn(inputs, labels, alpha=1.0)
    outputs = model(inputs)
    loss = lam * criterion(outputs, la) + (1 - lam) * criterion(outputs, lb)
    preds = outputs.argmax(1)
    correct = (preds == la).sum().item()
else:
    outputs = model(inputs)
    loss = criterion(outputs, labels.long())
    preds = outputs.argmax(1)
    correct = (preds == labels).sum().item()
```

Listing 7: CutMix loss calculation in training loop from `mycode/train.py`

3.1.5 Variant 5: Combined Attention and Regularization (CBAM + Dropout)

CBAM enables the model to identify and concentrate on the most discriminative features and Dropout provides a complementary benefit by preventing the model from becoming overly dependent on any specific subset of features or neurons. This combination gives a potential limitation of attention mechanisms. The risk that focused attention might lead to overfitting if the model becomes too specialized on particular feature patterns present in the training set but absent in the test set. I introduce controlled randomness that forces the model to maintain multiple pathways to the correct decision. **Detailed Methodology.** This variant combines the architectural modifications from Variant 3 (CBAM integration) with an additional regularization component in the classification pathway. The CBAM module is implemented exactly as described in Variant 3, providing both channel and spatial attention mechanisms. The regularization component consists of a Dropout layer with probability p=0.5, strategically placed in the classification head immediately before the final linear layer. During training, Dropout randomly sets 50% of the input features to zero, forcing the remaining features to compensate and preventing the co-adaptation of neurons. This regularization technique is particularly effective when combined with attention mechanisms, as it prevents the model from over-relying on the specific features highlighted by the attention module.

3.2 Results and Comparative Analysis

3.2.1 Quantitative Performance Comparison

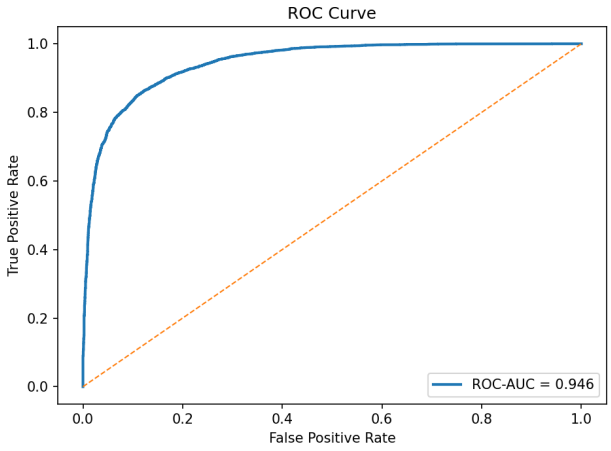
The evaluation protocol was identical across all variants to ensure fair comparison, with each model making predictions on the same 110,905 test images. The results, presented in Table 1, reveal several important insights about the relative effectiveness of different training strategies and architectural modifications.

Table 1: Performance comparison of supervised model variants on the test set. The best result for each metric is highlighted in bold.

Model Variant	Accuracy	F1-Score	ROC-AUC	PR-AUC
Baseline (from Scratch)	85.04%	0.8358	0.9460	0.9440
Transfer Learning	84.40%	**0.8645**	0.9428	0.9114
CBAM (from Scratch)	86.16%	0.8545	0.9434	0.9414
CutMix (from Scratch)	86.26%	0.8572	0.9392	0.9464
CBAM + Dropout	**86.92%**	0.8568	**0.9616**	**0.9653**

3.2.2 Analysis of Baseline Performance.

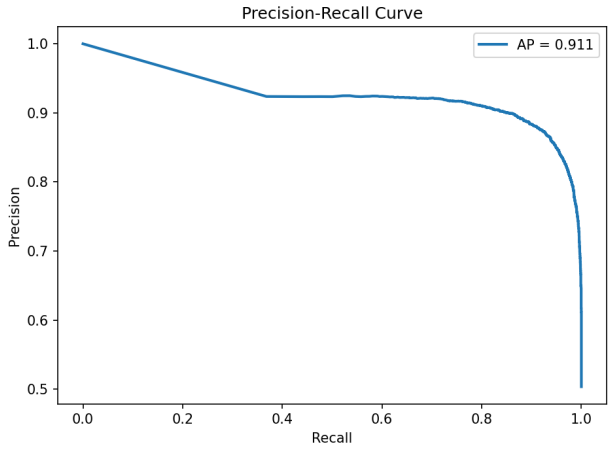
The baseline model trained from scratch (Variant 1) established a remarkably strong foundation, achieving 85.04% accuracy and a ROC-AUC of 0.9460. This performance level demonstrates that the deepfake dataset contains sufficient discriminative information for a deep convolutional network to learn effective feature representations without external knowledge. The high ROC-AUC value indicates excellent ranking performance, suggesting that the model successfully learned to assign higher confidence scores to fake images and lower scores to real images, even when the specific decision threshold at 0.5 may not be optimal.



3.2.3 Transfer Learning Results and Implications

Contrary to conventional expectations, transfer learning variant did not outperform the baseline in terms of overall accuracy (84.40% vs 85.04%) or ROC-AUC (0.9428 vs 0.9460). However, it achieved the highest F1-Score (0.8645), indicating a superior balance between precision and recall at the 0.5 decision threshold. This result suggests that while ImageNet pre-training may not provide additional discriminative power for this specific dataset, it does lead to better-calibrated predictions that align more closely with the default

threshold. The relatively modest benefit of transfer learning can be attributed to several factors. First, the deepfake dataset is substantial enough (290,335 images) to support effective learning from scratch. Second, the visual patterns characteristic of deepfake artifacts may be sufficiently distinct from natural image statistics that pre-trained ImageNet features, while useful for general object recognition, do not provide a significant advantage for detecting synthetic facial manipulations.



3.2.4 Impact of Attention Mechanisms

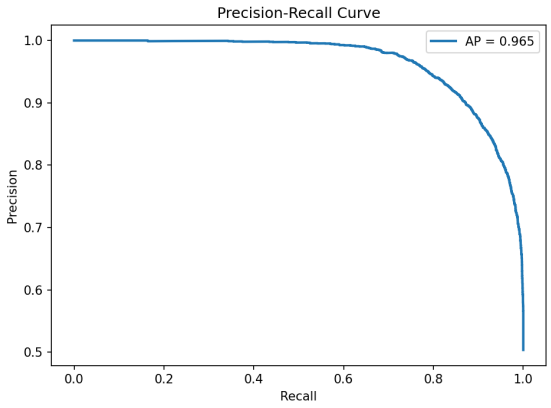
The CBAM-augmented model demonstrated a clear improvement over the baseline, achieving 86.16% accuracy—a gain of 1.12 percentage points. This improvement validates the hypothesis that explicit attention mechanisms can enhance deepfake detection by enabling the model to focus on the most informative facial regions and feature channels.

3.2.5 Advanced Regularization Effects

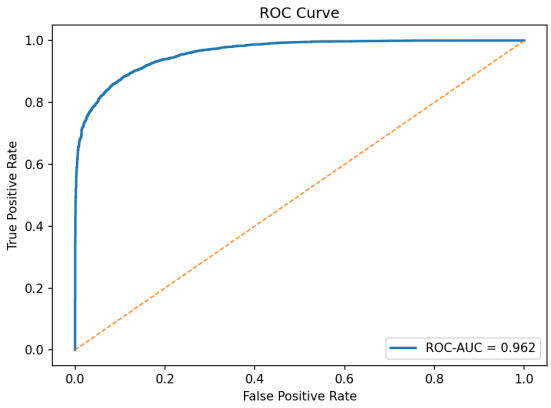
The CutMix variant achieved the second-highest accuracy among the from-scratch models (86.26%), surpassing both the baseline and CBAM variants. Tt also achieved the highest PR-AUC (0.9464) among the single-technique variants, suggesting that the regularization provided by training on composite images improved the model's precision-recall characteristics. This result supports the hypothesis that exposure to spatially mixed training examples forces the model to learn more distributed and robust feature representations.

3.2.6 Synergistic Effects of Combined Techniques

The most compelling results were achieved by the CBAM + Dropout variant (Variant 5), which emerged as the clear winner across multiple key metrics. With an accuracy of 86.92%, ROC-AUC of 0.9616 and PR-AUC of 0.9653, this model demonstrated the best overall performance.CBAM enables the model to identify and focus on the most relevant features, while Dropout prevents overfitting to those same features, resulting in improved generalization.



(a) PR Curve



(b) ROC Curve

3.2.7 Interpretability Analysis with Grad-CAM

To understand the decision-making process of models, I employed Gradient-weighted Class Activation Mapping (Grad-CAM), a technique that produces visual explanations for deep learning model predictions. Grad-CAM generates heatmaps that highlight the regions of the input image that most strongly influence the model's decision. [12]

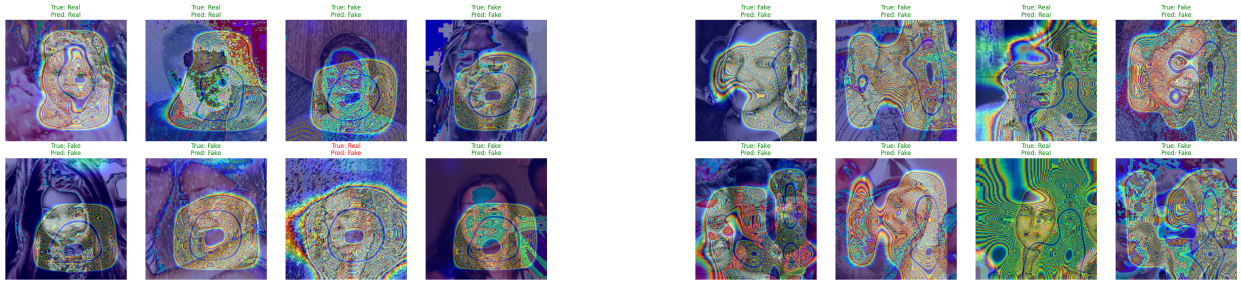
Methodology Grad-CAM works by computing the gradients of the class score with respect to the feature maps of the final convolutional layer. These gradients are global-average-pooled to obtain the neuron importance weights, which are then used to produce a localization map highlighting the important regions in the image. The implementation automatically selects the appropriate target layer. Specifically, the `conv2` layer of the final BasicBlock in `layer4`, with intelligent handling for models containing CBAM modules.

```
# Automatically select the final convolutional layer of the model as the target
layer4_children = list(model.layer4.children())
target_block = layer4_children[-1]
if target_block.__class__.__name__.lower() == "cbam":
    target_block = layer4_children[-2]  # Skip CBAM, use previous block
if not hasattr(target_block, "conv2"):
    raise RuntimeError("Can't find 'conv2' in the target block for Grad-CAM")
target_layer = target_block.conv2
cam_extractor = GradCAM(model, target_layer=target_layer)

# Generate activation map for predicted class
cam = cam_extractor(class_idx=pred, scores=out)[0].detach().cpu()
# Normalize for visualization
cam = (cam - cam.min()) / (cam.max() - cam.min() + 1e-6)
```

Listing 8: Grad-CAM target layer selection from `mycode/cam_vis.py`

Analysis of Activation Patterns. The Grad-CAM analysis reveals that models have learned to focus on semantically meaningful facial regions consistent with domain knowledge about deepfake artifacts. For authentic images, models typically concentrate attention on natural facial features such as eyes, nose, and mouth with activation patterns following natural contours and texture variations. For synthetic images, the models show concentrated attention on regions known to be challenging for deepfake generation algorithms: eye regions (iris details, eyelid boundaries), mouth and lips (teeth visibility, texture inconsistencies), face-background boundaries and skin texture regions where generation artifacts commonly appear.



4 Approach 2: Anomaly Detection via Adversarial Training

The second methodology investigates anomaly detection using Generative Adversarial Networks (GANs). Unlike supervised learning, this approach learns the underlying distribution of authentic facial images and identifies deviations from this learned normality as potential deepfakes. [16]

This approach is motivated by the hypothesis that if a model comprehensively understands authentic human faces, it should identify synthetic images as statistical outliers. Practically, anomaly detection may be more robust to evolving deepfake techniques as it relies on understanding fundamental characteristics of authentic imagery rather than learning artifacts of specific generation methods. [16]

4.1 Methodology

4.1.1 Conceptual Framework

I leveraged the discriminator component of a GAN as an "expert in authenticity." The discriminator learns to distinguish between real images from the training distribution and synthetic images produced by the generator. I trained exclusively on authentic facial images, making the discriminator exceptionally skilled at recognizing authentic faces.

After training, the discriminator outputs a "realness" score for test images. High scores indicate authentic images, while low scores flag potential deepfakes. The decision boundary emerges from the discriminator's internalized model of authentic facial imagery, without explicit exposure to negative examples.

4.1.2 GAN Architecture

Implementation follows Deep Convolutional GAN (DCGAN) principles, adapted for high-resolution facial image processing. The generator transforms random latent vectors into realistic facial images through transposed convolutions, while the discriminator distinguishes between generated and authentic training images. [19]

```
class Generator(nn.Module):
    def __init__(self, latent_dim=100, channels=3):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            # input is Z, going into a convolution that creates a 7x7 base
            nn.ConvTranspose2d(latent_dim, 1024, 7, 1, 0, bias=False),
            nn.BatchNorm2d(1024),
            nn.ReLU(True),
            # state size. 1024 x 7 x 7
            nn.ConvTranspose2d(1024, 512, 4, 2, 1, bias=False),
```

```
            nn.BatchNorm2d(512),
            nn.ReLU(True),
            # state size. 512 x 14 x 14
            nn.ConvTranspose2d(512, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.ReLU(True),
            # state size. 256 x 28 x 28
            nn.ConvTranspose2d(256, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(True),
            # state size. 128 x 56 x 56
            nn.ConvTranspose2d(128, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            # state size. 64 x 112 x 112
            nn.ConvTranspose2d(64, channels, 4, 2, 1, bias=False),
            nn.Tanh()
            # final state size. channels x 224 x 224
        )

    def forward(self, input):
        return self.main(input)
```

Listing 9: Generator architecture implementation from `mycode/gan_models.py`

The generator transforms a 100-dimensional latent vector (sampled from standard normal distribution) through five transposed convolutional layers with batch normalization and ReLU activation. The progressive upsampling ($7\times7 \rightarrow 14\times14 \rightarrow 28\times28 \rightarrow 56\times56 \rightarrow 112\times112 \rightarrow 224\times224$) builds facial features hierarchically, from overall structure to fine-grained details. The final layer uses Tanh activation to constrain outputs to [-1, 1], matching the normalization scheme of training images.

Discriminator Architecture The discriminator performs the complementary operation taking a 224×224 RGB image as input and producing a single logit representing the "realness" score. The architecture employs six convolutional layers with LeakyReLU activation and batch normalization, progressively downsampling the spatial dimensions ($224\times224 \rightarrow 112\times112 \rightarrow 56\times56 \rightarrow 28\times28 \rightarrow 14\times14 \rightarrow 7\times7$) while increasing channel depth ($3 \rightarrow 64 \rightarrow 128 \rightarrow 256 \rightarrow 512 \rightarrow 1024$). The final 1×1 feature map is flattened to produce the output score, with no sigmoid activation since BCEWithLogitsLoss is used for numerical stability. The discriminator employs LeakyReLU activations (negative slope 0.2) to prevent gradient sparsity during adversarial training. Batch normalization is applied after each convolutional layer except the first and last to stabilize training. The final output is a single logit that represents the "realness" score, with sigmoid applied only during evaluation to obtain probability values.

4.1.3 Adversarial Training Process

The training process follows standard GAN optimization but uses exclusively authentic facial images. Dataset consists of 70,001 real images from the training set, ensuring the discriminator develops expertise in recognizing natural facial characteristics without exposure to deepfake artifacts.

Training Configuration Both networks use Adam optimizers (lr=0.0002, betas=(0.5, 0.999)) with BCEWithLogitsLoss for numerical stability. The adversarial objective trains the discriminator to distinguish between real dataset images and generator-produced images, while the generator learns to fool the discriminator.

```python
# Filter dataset to only Real images
real_indices = [i for i, (_, label) in enumerate(dataset.samples) if label == 0]
real_dataset = torch.utils.data.Subset(dataset, real_indices)

# Standard GAN training loop
for epoch in range(args.epochs):
    for i, data in enumerate(dataloader):
        # Train Discriminator on Real vs Generated
        real_cpu = data[0].to(device)
        errD_real = criterion(netD(real_cpu).view(-1), real_label)

        noise = torch.randn(batch_size, args.latent_dim, 1, 1, device=device)
        fake = netG(noise)
        errD_fake = criterion(netD(fake.detach()).view(-1), fake_label)
        errD = errD_real + errD_fake
        optimizerD.step()

        # Train Generator to fool Discriminator
        errG = criterion(netD(fake).view(-1), real_label)
        optimizerG.step()
```

Listing 10: Key components of adversarial training from `train_gan.py`

Training Dynamics Training for 50 epochs revealed typical GAN dynamics with alternating network dominance. Early epochs showed balanced competition (Loss_D ≈ 0.7, Loss_G ≈ 3.8), followed by phases where the discriminator achieved near-perfect classification (Loss_D < 0.01) while the generator struggled (Loss_G > 10). This pattern indicates successful discriminator learning. It developed strong expertise in recognizing authentic faces, which is essential for effective anomaly detection during testing.

4.2 Results and Analysis

4.2.1 Quantitative Performance

The GAN discriminator was evaluated on the test set on 70,001 authentic images.

Table 2: Performance comparison between supervised and GAN-based approaches.

Approach	Accuracy	F1-Score	ROC-AUC	EER
CBAM + Dropout (Supervised)	**86.16%**	**0.8545**	**0.9434**	**0.1325**
GAN Discriminator (Anomaly)	47.20%	0.4112	0.4701	0.5217

The GAN achieved 47.20% accuracy (below random chance) with ROC-AUC of 0.4701, indicating worse-than-random performance. The EER of 0.5217 confirms no effective decision threshold exists.

4.2.2 Training Dynamics Analysis

Training exhibited unstable GAN dynamics with discriminator dominance:

```
Epoch 1:  Loss_D=0.7123, Loss_G=3.7792   # Initial balance
Epoch 23: Loss_D=0.0010, Loss_G=10.3531  # D near-perfect
Epoch 46: Loss_D=0.0012, Loss_G=35.8950  # G collapse
Epoch 50: Loss_D=0.2770, Loss_G=9.8736   # Final state
```

Listing 11: Key training loss evolution

The discriminator rapidly achieved near-zero loss, providing minimal gradient signal for generator learning. Generator loss spiked to 35.9, indicating training collapse. The results highlight fundamental limitations of GAN-based anomaly detection for deepfakes. Modern synthetic images are too realistic to be detected as statistical outliers, and training instability prevents effective learning of authentic facial characteristics without explicit negative examples.

5 Conclusions

This comparative investigation demonstrates a fundamental asymmetry in the effectiveness of supervised versus unsupervised learning paradigms for deepfake detection. The empirical evidence reveals that supervised classification substantially outperforms anomaly detection approaches, with best-performing ResNet-18 variant achieving 86.16% accuracy compared to the GAN discriminator's 47.20% performance. Several promising avenues emerge from this work.

- **Architectural innovations** could explore transformer-based attention mechanisms beyond CBAM, potentially incorporating self-attention across temporal dimensions for video deepfake detection. **Hybrid methodologies** might combine supervised classification with contrastive learning, enabling models to learn robust feature representations while maintaining discriminative capability.
- **Architectural innovations** could explore transformer-based attention mechanisms beyond CBAM, potentially incorporating self-attention across temporal dimensions for video deepfake detection. **Hybrid methodologies** might combine supervised classification with contrastive learning, enabling models to learn robust feature representations while maintaining discriminative capability.
- **Dataset diversification** represents a critical research need. Future studies should evaluate performance across multiple generation techniques (StyleGAN, Stable Diffusion, commercial deepfake tools) and temporal consistency in video sequences. **Adversarial robustness** investigations could assess model performance against dedicated evasion attacks designed to exploit learned features.
- **Adversarial robustness** investigations could assess model performance against dedicated evasion attacks designed to exploit learned features.
- **Novel anomaly detection approaches** merit continued exploration. Variational autoencoders, normalizing flows, or diffusion-based density models might overcome the training instabilities that limited GAN implementation. [18]
- **Ensemble methodologies** combining multiple detection paradigms could potentially leverage the complementary strengths of different approaches.

The development of robust detection mechanisms becomes not merely a technical challenge, but a fundamental prerequisite for preserving the integrity of visual truth in an increasingly synthetic world.

Bibliography

[1] Jia Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009). Referenced in dataset.py - transfer learning foundation, pp. 248–255. URL: https://ieeexplore.ieee.org/document/5206848.

[2] François-Guillaume Fernandez. *TorchCAM: Class Activation Explorer*. Referenced in cam_vis.py - GradCAM implementation. 2020. URL: https://github.com/frgfm/torch-cam.

[3] Ian Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems* (2014). Referenced in train_gan.py - foundational GAN theory, pp. 2672–2680. URL: https://arxiv.org/abs/1406.2661.

[4] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv preprint arXiv:1512.03385* (2015). Referenced in cam_vis.py - foundational CNN architecture. URL: https://arxiv.org/abs/1512.04150.

[5] *Instagram Reel: DKZ6qdqSl0L*. Social media reference - deepfake detection content. 2024. URL: https://www.instagram.com/reel/DKZ6qdqSl0L/?igsh=Y28zMW9jcGphenc3.

[6] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv preprint arXiv:1502.03167* (2015). Referenced in gan_models.py and model.py - normalization technique. URL: https://arxiv.org/abs/1502.03167.

[7] Manjil Karki. *DeepFake and Real Images Dataset*. Referenced throughout project - primary dataset used. 2023. URL: https://www.kaggle.com/datasets/manjilkarki/deepfake-and-real-images.

[8] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv preprint arXiv:1412.6980* (2014). Referenced in train.py and train_gan.py - optimization algorithm. URL: https://arxiv.org/abs/1412.6980.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems* 25 (2012). Referenced in dataset.py - foundational CNN architecture, pp. 1097–1105. URL: https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[10] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems* (2019). Referenced throughout project - main deep learning framework, pp. 8024–8035. URL: https://arxiv.org/abs/1912.01703.

[11] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *arXiv preprint arXiv:1511.06434* (2015). Referenced in gan_models.py - GAN architecture implementation. URL: https://arxiv.org/abs/1511.06434.

[12] Ramprasaath R. Selvaraju et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization". In: *arXiv preprint arXiv:1610.02391* (2016). Referenced in cam_vis.py - model interpretability technique. URL: https://arxiv.org/abs/1610.02391.

[13] PyTorch Team. *TorchVision: PyTorch's Computer Vision Library*. Referenced in model.py - pre-trained model library. 2016. URL: https://pytorch.org/vision/stable/index.html.

[14] Cigdem Beyan Vittorio Murino. *Attention Mechanisms and Transformers*. PDF slides: attentionMec_Transformers.pdf. Course material on attention mechanisms and transformers. 2025.

[15] Cigdem Beyan Vittorio Murino. *Deep Learning Classification*. PDF slides: 2-Classification.pdf. Course material on deep learning classification techniques. 2025.

[16] Cigdem Beyan Vittorio Murino. *Generative Models*. PDF slides: GenerativeModels.pdf. Course material on generative models including GANs. 2025.

[17] Cigdem Beyan Vittorio Murino. *LAB_02: CNN Exercise*. Jupyter Notebook: lab02b CNN_exercise.ipynb. Laboratory exercise on Convolutional Neural Networks. 2025.

[18] Cigdem Beyan Vittorio Murino. *LAB_08: Denoising Autoencoder and VAE*. Jupyter Notebook: LAB_08.ipynb. Laboratory exercise on autoencoders and variational autoencoders. 2025.

[19] Cigdem Beyan Vittorio Murino. *LAB_09: DCGAN on EMNIST*. Jupyter Notebook: LAB_09.ipynb. Laboratory exercise on Deep Convolutional Generative Adversarial Networks. 2025.

[20] Sanghyun Woo et al. "CBAM: Convolutional Block Attention Module". In: *arXiv preprint arXiv:1807.06521* (2018). Referenced in cbam.py - attention mechanism implementation. URL: `https://arxiv.org/abs/1807.06521`.

[21] Sangdoo Yun et al. "CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features". In: *arXiv preprint arXiv:1905.04899* (2019). Referenced in augment.py - data augmentation technique. URL: `https://arxiv.org/abs/1905.04899`.