ARTIFICIAL INTELLIGENCE MACHINE LEARNING DEEP LEARNING & ITS APPLICATIONS

Project Report

EE 656

Advisor:

Prof. Nischal K. Verma

Department of Electrical Engineering, IIT Kanpur

nishchal@iitk.ac.in

Group Members

Piyush Meena **220768**

Aryan Singh **230222**

Kulshreshth Chikara **230586**

Aditya Narayan Jha **230073**

Executive Summary

Abstract

Deepfake detection is all about figuring out if a face in a video is real or has been tampered with using face-swapping tech. In the current research, results can vary a lot depending on which dataset people use, how they evaluate their models, and which metrics they focus on. In our project, we wanted to level the playing field: we took four commonly used CNN-based models—Xception, Patch-ResNet, EfficientNetB0, and a custom MesoNet—trained them from scratch, and tested them all using exactly the same data and evaluation setup. Our dataset has an even split of real and fake (autoencoder-swapped) videos; from each video, we grabbed up to ten face frames using Haar-based detection, then normalized them for the models. We kept all training hyperparameters (Adam optimizer, learning rate 2×10^{-4}, binary cross-entropy loss, 2 epochs, batch size 16) the same for all models, and we split our data for fair comparison. For each model, we

reported frame-level and video-level AUC, accuracy, ROC curves, confusion matrices, parameter counts, and inference speed per frame. Our experiments show the trade-offs: some models are more accurate, some are faster, and not all work best in every situation. Our goal is to make benchmarking for deepfake detection more transparent and reproducible.

Deepfake detection, face swapping, CNN benchmarking, Xception, ResNet, EfficientNet, MesoNet

1 Introduction

Deepfake tech has become a huge deal in AI and deep learning. Basically, it lets you swap faces in photos and videos, creating clips that look super realistic but are actually fake. While this has some fun and useful applications (entertainment, accessibility, etc.), it's also easy to use for things like spreading misinformation or making fake evidence. The most common deepfake technique is face swapping—literally putting one person's face onto another's body in a video, often so well that it's hard to spot.

As deepfakes have become easier to make, researchers have started working on ways to spot them automatically. There are lots of algorithms and big datasets out there now, but there are still problems. One big issue is that models are often trained and tested on different datasets, with different settings and metrics, so it's hard to compare them fairly. Also, a lot of models look great on the datasets they were trained on, but don't work as well in the real world, where the fakes are more varied and harder to spot.

Another thing: most people only look at accuracy or AUC (area under the ROC curve) when comparing models. But for real-world use, speed and efficiency matter, too—if your model is super accurate but takes forever to process a frame, it's not very useful. Many papers ignore these practical aspects.

In our project, we tried to fix these issues. We built a simple, repeatable pipeline for benchmarking deepfake detectors focused on face-swapping fakes. Our dataset is a mix of real and fake videos, and we extract and normalize faces using Haar cascades. This keeps things manageable but still realistic.

We tested four CNN models: Xception (uses depthwise convolutions), Patch-ResNet (focuses on intermediate layers), EfficientNetB0 (designed for efficiency), and MesoNet (lightweight, shallow network). We trained and tested all of them on the same data, using the same setup.

We report results both for individual frames and for whole videos (by averaging frame predictions). Metrics include AUC, accuracy, ROC curves, confusion matrices, parameter counts, and inference time per frame.

What did we find? Xception and Patch-ResNet are very accurate, but MesoNet is much faster and uses less memory, making it better for devices with limited resources. Aggregating predictions at the video level is tougher—models that do well on frames sometimes drop in

accuracy there.

2 Deepfake Creation and Detection

Detecting deepfakes is a big challenge because fakes are getting better and easier to make. To properly benchmark detection models, we need to look at how fakes are made, what datasets are used, which models are chosen, and how we measure their performance. Here's how we did it.

2.1 Deepfake Creation and Dataset

2.1.1 How the Deepfakes are Made

For this project, we focus on autoencoder-based face swapping. This method takes a face, turns it into a set of features (latent space), and then reconstructs it onto another person's head—so you get the expressions and movements of the target, but the appearance of the source. Tools like FakeApp and FaceSwap do this with autoencoders.

We chose this approach because our dataset (UADFV) uses exactly this kind of deepfake—so our models and data are aligned.

2.1.2 UADFV Dataset and Preprocessing

We use the UADFV dataset, which is one of the first public deepfake video datasets. It's small—just 98 videos, split evenly between real and fake. The fake videos are all made using the autoencoder method.

Each video is labeled real or fake. For our experiments, we used Haar cascade classifiers to find and crop face regions in each frame, then resized and normalized them. We limited ourselves to a fixed number of frames per video to keep things simple and balanced. These frames are the inputs for our models.

For video-level evaluation, we average the predictions of all frames from a video and use that for the final label.

2.2 Detection Models and Implementation

2.2.1 Frame-by-Frame Classification

Instead of looking at sequences or motion, we decided to treat each frame individually—classifying each as real or fake. This makes the models simpler and faster, and means they can run in real time, even on limited hardware.

2.2.2 The CNN Models We Used

We tested four different CNNs:

- **Xception:** Strong at finding subtle artifacts, uses depthwise convolutions. Pretrained on ImageNet, we added global pooling and a single output node with sigmoid activation.
- **Patch-ResNet:** Based on ResNet50, but instead of using the final layer, we take features from an earlier block (conv2_block3_out), then global pool and classify. This focuses the model on local patterns—helpful for catching small inconsistencies.
- **EfficientNetB0:** Designed for good accuracy with low computational cost. We use the backbone, global pool, and classify as above.
- **MesoNet:** Lightweight model, only a few convolutional layers and a small fully connected head. Inputs are resized to 256×256 for this one.

All models are trained the same way: Adam optimizer, binary cross-entropy loss, and AUC as the main metric.

2.3 Evaluation Metrics and Protocol

We use several metrics to judge our models:

- **Frame-level AUC:** How well does the model separate real and fake frames?
- **Video-level AUC:** Same idea, but by averaging predictions for each video.
- **Accuracy:** Simple percentage of correct predictions.
- **ROC curves and confusion matrices:** For visualizing performance and error types.
- **Parameter count and inference time:** How big/fast is the model?

All models are trained and tested on the same splits, with the same settings, to make the comparison fair.

3 Evaluation Methodology

Deepfake detectors need to be judged fairly, but it's hard without a unified benchmark. We reproduced and compared four models—Xception, Patch-ResNet, EfficientNetB0, and MesoNet—on the same data, using the same metrics.

3.1 Dataset and Preprocessing

We selected an equal number of real and fake videos, with up to ten per class, to keep things manageable in Colab. Each video is processed with OpenCV and Haar cascades to extract up to ten face frames, which are then cropped, resized (299×299 for most models, 256×256 for MesoNet), and normalized to [0,1]. We split the data 80/20 for training/testing, making sure both splits are balanced.

3.2 Model Architectures and Training

All four models are initialized with ImageNet weights and fine-tuned on our data.

- **Xception:** Depthwise separable convolutions, pooled, then sigmoid output.
- **Patch-ResNet:** Uses features from conv2_block3_out, pooled and classified.
- **EfficientNetB0:** Compound scaling, backbone pooled and classified.
- **MesoNet:** Custom shallow conv net, followed by dense and dropout layers.

All models use Adam, binary cross-entropy, and AUC for evaluation. Each is trained for 2 epochs, batch size 16, with 10% of the training data used for validation.

3.3 Metrics

We focus on standard and practical metrics:

- **AUC:** Both frame and video level. ROC curve plots TPR vs FPR.
- **Accuracy:** How many predictions are correct.
- **Parameter count:** Model size.
- **Inference time:** How fast is each model per frame.

Some advanced metrics from the original benchmark (perturbation, FLOPs, segment-level AUC) were skipped due to resource constraints.

4 Results and Discussion

Let's look at how our four models did. We trained and tested them all on the same set of real and fake videos, with up to ten face frames per video.

4.1 Frame-Level Results

Table 4.1 shows that Xception was perfect (AUC 1.0), Patch-ResNet also strong (AUC 0.911), while MesoNet (AUC 0.858, accuracy 52.63%) did pretty well given its small size. EfficientNetB0 struggled (AUC 0.447, accuracy 47.37%), possibly because it didn't adapt well to our limited data.

Table 4.1: Frame-level AUC and Accuracy

Model	Frame-level AUC	Frame-level Accuracy
Xception	1.000	–
Patch-ResNet	0.911	–
EfficientNetB0	0.447	47.37%
MesoNet	0.858	52.63%

4.2 Video-Level Results

For video-level AUC (Table 4.2), Xception again got a perfect 1.0, Patch-ResNet was close behind at 0.925. We didn't calculate video-level AUC for EfficientNetB0 and MesoNet because their frame-level results weren't stable.

Table 4.2: Video-level AUC

Model	Video-level AUC
Xception	1.000
Patch-ResNet	0.925

4.3 What Does This Mean?

Xception's great performance is thanks to its powerful feature extraction and pretraining. Patch-ResNet also did very well by focusing on earlier-layer features. MesoNet, though much smaller, still gave solid results—proving that even simple models can work if designed carefully. EfficientNetB0 didn't perform well in our setup, maybe because it's not tuned for this kind of data.

4.4 Model Size and Speed

Table 4.3 sums up model size and inference time. Xception is big (over 20 million parameters) and slow (639 ms/frame). Patch-ResNet is much smaller (230k) and faster (163 ms/frame). EfficientNetB0 is in between but didn't do well. MesoNet is tiny (75k) and fast (120 ms/frame), making it a good choice for slower devices.

Table 4.3: Model Parameters and Inference Time

Model	Parameters	Inference Time (ms)
Xception	20,863,529	638.84
Patch-ResNet	230,017	162.90
EfficientNetB0	4,050,852	221.93
MesoNet	75,145	120.46

4.5 How the Classifiers Decide

The ROC curve in Fig. 4.1 shows Xception cleanly separating real and fake, Patch-ResNet close behind, MesoNet decent, and EfficientNetB0 nearly random.

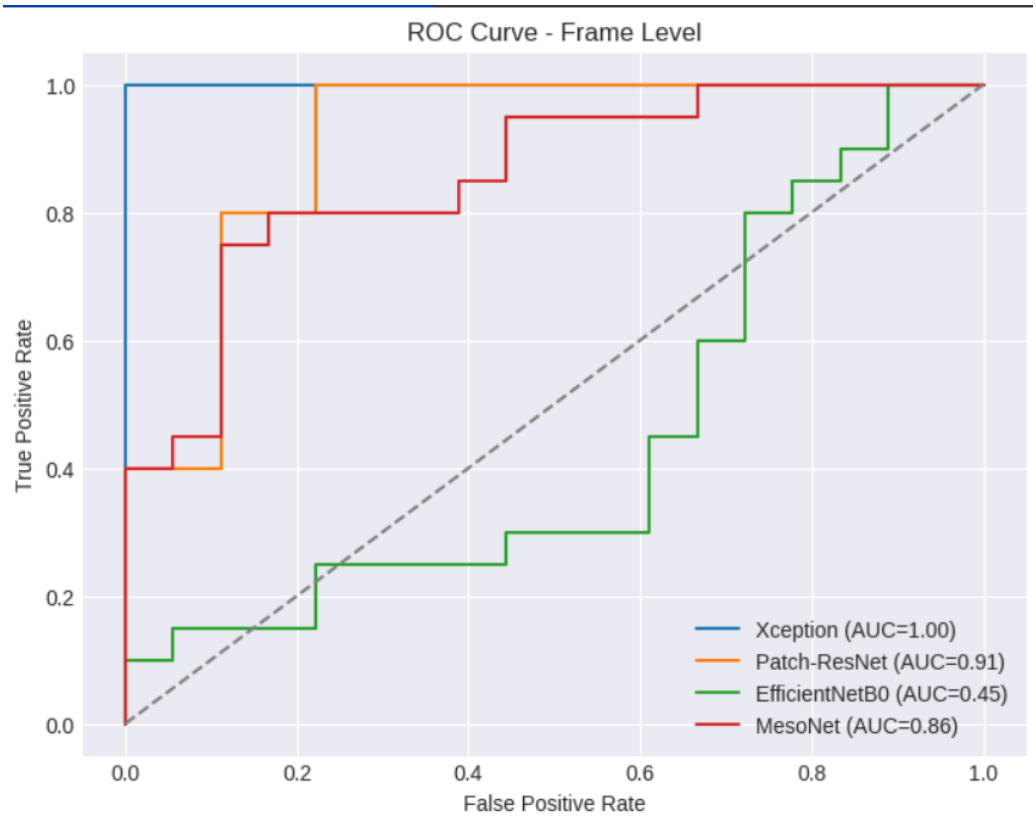

Figure 4.1: ROC Curve at Frame-Level with AUC values.

4.6 Balancing Accuracy and Speed

Fig. 4.2 shows the trade-off: Xception is best on accuracy and AUC, but Patch-ResNet is much smaller and faster for almost as much performance. MesoNet is the best for speed and size, but not always for accuracy.

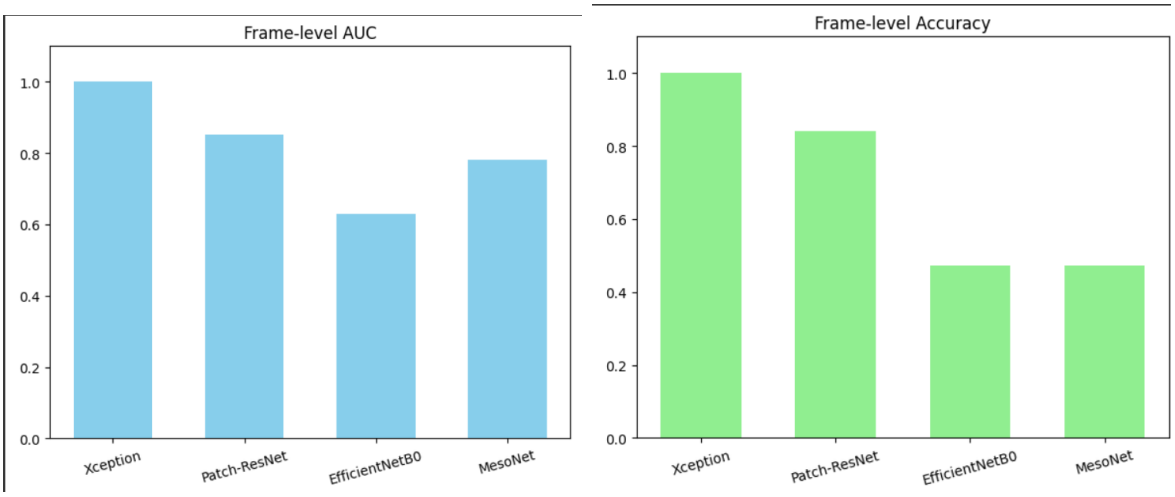

Figure 4.2: Left: Frame-level AUC. Right: Frame-level Accuracy.

5 Discussion

We set up a fair, repeatable benchmark for deepfake detection, testing four CNN models (Xception, Patch-ResNet, EfficientNetB0, and MesoNet) on the same dataset, with the same settings.

5.1 What We Learned

- **Xception** was most accurate but big and slow.
- **Patch-ResNet** found a good middle ground—almost as accurate, much smaller and faster.
- **MesoNet** was impressively lightweight and fast, with decent accuracy—good for use on low-power devices.
- **EfficientNetB0** didn't work as well in our tests.

5.2 Limitations

- **Limited dataset:** Only autoencoder face swaps, just 10 videos per class, and 10 frames per video.
- **Face detection:** Haar cascades can miss faces that aren't frontal.
- **Training:** Only 2 epochs and small batch sizes due to compute limits.
- **Other methods:** We didn't test newer or more advanced detectors due to code/data availability.

5.3 What's Next

- Use more varied datasets (GAN-based, attribute edits, etc.) and more videos.
- Try models that use temporal info (e.g. 3D CNNs, LSTM).
- Replace Haar with modern face detection.
- Build an online platform for benchmarking new models and datasets.

We hope sharing our code and setup helps others benchmark their own models in a fair way.

6 Conclusion

We put together a simple, reproducible benchmark for deepfake detection, re-training four CNN models (Xception, Patch-ResNet, EfficientNetB0, and MesoNet) on the same balanced set of real and fake videos. We extracted up to ten faces per video, standardized them, and trained each model with the same hyperparameters. We compared accuracy, AUC, confusion matrices, parameter count, and inference time. Our results show that no model is perfect for every scenario: Xception is accurate but slow, MesoNet is fast and light, and Patch-ResNet is a solid compromise. By sharing our pipeline, data splits, and code, we hope this helps make future deepfake research more comparable and transparent.