Model Challenge 2

Deepfake Duel: Truth vs. Trickery

Team Shamrock – **Anuva Negi, Shreyas Rajapur Sanjay, Smrithi Chembath**

Abstract :

This project aims to tackle the growing concern of deepfake imagery by building a machine learning-based web application capable of classifying images as **real** or **fake**, and further categorizing them into domains such as **human faces**, **vehicles**, or **animals**. The application is built using a PyTorch-trained model, deployed via Flask, and hosted on Render with a custom domain for public access.

Methodology

The solution follows an end-to-end pipeline comprising **model training**, **Flask-based app development**, and **deployment with a custom domain**. The workflow was divided into the following stages:

1. Dataset and Preprocessing

We used a subset of the ArtiFact_240K dataset that included labelled examples of real and fake images across three major categories: **animals**, **vehicles**, and **human faces**. Each image was labelled with a binary indicator (`0` for fake, `1` for real) and classified into one of the three domains. Images were resized and normalized using standard PyTorch transforms, with augmentation strategies like random flips and rotations to enhance model generalization.

2. Model Architecture

We fine-tuned a **XceptionNet** architecture, pretrained on ImageNet. The final fully connected layer was modified to output both:

- A binary prediction (real or fake)
- A multiclass classification (category of the image)

The model was trained using **Binary Cross-Entropy** loss for real/fake prediction and **Cross-Entropy** loss for class category, using a weighted combination to optimize performance. We used the **Adam** optimizer with early stopping based on validation loss.

3. Web Application

The user interface was built using **html** and **Flask**, allowing users to upload images, which are then processed in real time. Uploaded images are saved temporarily and passed through the model. The prediction results are displayed on a results page with labels like: “Fake” and “Human” for their respective results.

The backend loads the `.pt` model on startup, applies image transforms, and returns predictions using PyTorch inference.

4. Deployment

The app was deployed using **Render** and served via **python** for production readiness. A `requirements.txt` file was used to install dependencies, and a `Procfile` defined the entry point for Render:

```
web: python app:app
```

We connected the app to a custom domain: **thatlooksus.tech** and handled DNS configuration with both A and CNAME records pointing to Render's servers.

Readability and Interpretability of Model

- This model uses SmoothGradCAM++, an interpretability technique that highlights the most influential image regions driving the model's predictions.
- The classifier is a dual-headed Xception-based network trained to predict both the object class (Animal, Human, or Vehicle) and image authenticity (real or fake). Here, we focus on the classification head.
- Grad-CAM analyzes gradients from intermediate convolutional layers to reveal which image areas influenced the decision. Though no visual output is shown, the internal activation map confirms the model relies on specific spatial patterns.
- This enhances model transparency, helps detect overfitting, and supports explainability which is crucial for trustworthy AI in tasks like deepfake detection.

Challenges Encountered

1. **Git & File Tracking Errors:** While pushing the project to GitHub, we faced persistent issues where `requirements.txt` and `Procfile` were not being tracked or staged correctly, causing deployment failures on Render. These were resolved by manually removing cached Git entries and re-adding files.
2. **Render Deployment Errors:** Initial attempts to deploy resulted in 502 errors, largely due to incorrect references in the `Procfile` and a mismatch between the app filename and module name. Switching from `model_predict.py` to `app.py` required updating all internal references and the deployment command.
3. **Using Free Resources:** It came with its own shortcomings as we are not able to keep the website for a longer period of time
4. **Massive Dataset:** It's very difficult to load/extract/transform the data and also consuming significant amount of time.

Findings & Results

- The final model achieved an average accuracy of almost P70% on the validation set, with strong performance across all three domains.
- We trained on train test and cross- validated and tested it on Validation set in-order to understand the performance of the model. The test results are stored in test.csv as instructed
- The model was able to correctly identify deepfakes with consistent confidence, especially in the human face category.
- The full app is now live at https://www.thatlooksus.tech/ , offering public access with a simple, intuitive UI.

Conclusion

This project demonstrates how machine learning models can be integrated into real-world applications with a usable frontend and secure cloud deployment. It combines computer vision, software engineering, and domain knowledge to create a practical tool for deepfake detection. The application is scalable, extensible to more domains, and lays a solid foundation for further research or commercial deployment in media verification, journalism, or education.