DEEP LEARNING COURSE
PROJECT BASED LEARNING - SAMPLE PROJECT

Real-Time Deepfake Detection Using Convolutional Neural Networks

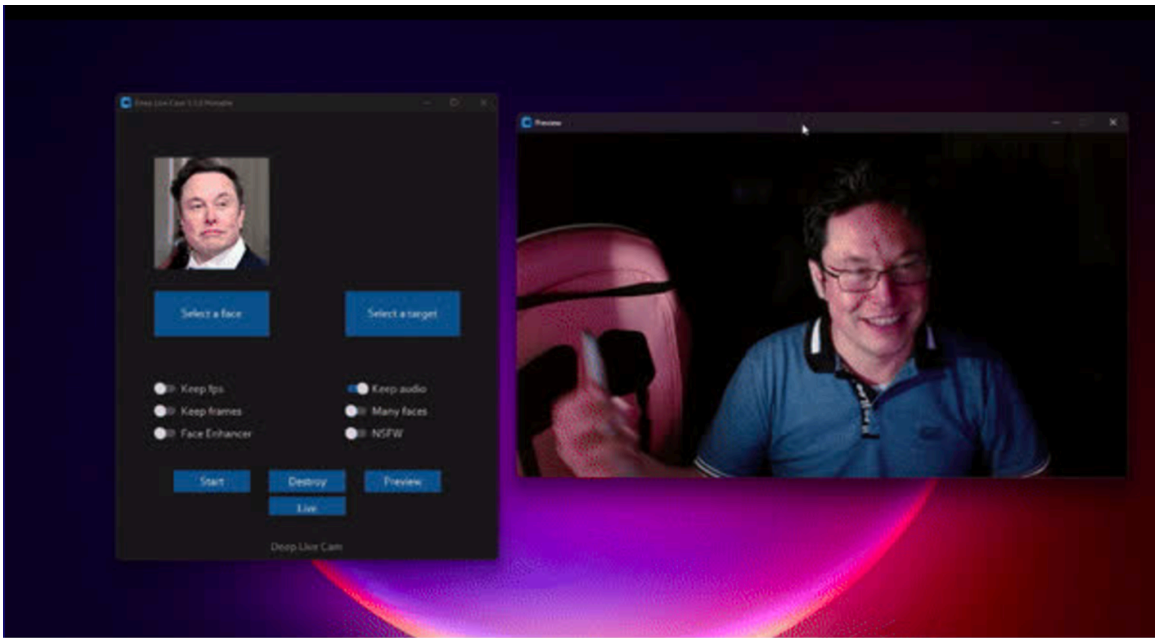
1. Objective

This project focuses on developing and deploying a deep learning-based system for detecting deepfakes in videos, addressing a critical challenge in artificial intelligence and cybersecurity. Students will apply their knowledge from the course to build a robust system capable of identifying manipulated video content in real time.

The project follows the complete deep learning pipeline as outlined in the syllabus:

1. **Data Preprocessing:** Working with datasets like FaceForensics++ and DFDC, extracting and preparing video frames.
2. **Model Development:** Designing CNN architectures, leveraging transfer learning, and using regularization and optimization techniques.
3. **Evaluation:** Assessing performance with metrics like accuracy, recall, and AUC-ROC.
4. **Deployment:** Integrating the trained model with OpenCV for real-time inference.

By completing this project, students will gain hands-on experience in designing, training, and deploying deep learning applications, aligning with the course objectives and preparing them for real-world AI challenges.



2. Tools Required

- Development Environment:
  - Jupyter Notebook or Google Colab.
  - Python programming language.
  - TensorFlow/Keras for model building and training.
- Dataset Management:
  - Kaggle or similar platforms for dataset access (e.g., FaceForensics++, DFDC).
  - Pandas, NumPy for data processing.
- Visualization and Metrics:
  - Matplotlib, Seaborn for plotting metrics and visualizations.
  - Scikit-learn for generating confusion matrices and other evaluation metrics.
- Real-Time System Integration:
  - OpenCV for webcam access and frame processing.
- Version Control:
  - Git/GitHub for collaboration and version tracking.
- Compute Resources:
  - Cloud platforms like Google Cloud, AWS, or Azure for training (72 hours allocated per group).
  - NVIDIA GPUs if available locally.

3. Learning Outcomes

- CLO1: Design and train efficient neural network architectures, understanding their key components and parameters.
- CLO2: Apply optimization algorithms, regularization techniques, and hyperparameter tuning to improve model performance.
- CLO3: Conduct error analysis, identify biases, and implement strategies for robust deep learning applications.
- CLO4: Develop convolutional neural networks for complex visual tasks and explore transfer learning for better generalization.
- CLO6: Execute a deep learning project from start to finish, including:
  - Data collection and preprocessing.
  - Model design, training, and evaluation.
  - Integration into a real-world application (real-time deepfake detection).

4. Deliverables

1. Preprocessed Data: Scripts and processed datasets.
2. Trained Model: Logs, saved model files, and performance metrics.
3. Evaluation Results: A detailed report of evaluation metrics.
4. System Integration: A fully functional real-time detection system.
5. Final Report: A technical document summarizing all steps and findings.
6. Presentation and Demo: A concise, clear explanation and demo.

5. Detailed 10-Week Plan

Week 1: Course Overview and Project Setup
Introduction to deep learning, project lifecycle, and team formation.
Tasks:
- Introduction to the importance of detecting deepfakes and societal implications.
- Explore publicly available datasets (e.g., FaceForensics++, DFDC).
- Assign groups (3-4 students per group) and distribute roles.
- Review example projects and tools for inspiration.

Week 2: Dataset Exploration and Preprocessing
Neural network basics and data preprocessing techniques.
Tasks:
- Select dataset(s) and understand its structure (real vs. fake videos).
- Write a script to extract frames from videos (e.g., one frame per second).
- Resize all frames to a uniform size (e.g., 128x128 or 256x256).
- Augment data with transformations (e.g., rotations, flips, noise addition).
- Balance the dataset to ensure even class representation.

Week 3: Building a Shallow Neural Network
Shallow neural networks as a baseline.
Tasks:
- Create a simple feedforward neural network to classify images.
- Use the preprocessed data to train the model as a baseline.
- Evaluate its performance (accuracy and loss) to highlight limitations.
- Begin planning a CNN model to overcome shallow NN limitations.

Week 4: Designing the CNN
Basics of convolutional neural networks (CNNs).
Tasks:
- Learn about convolutional layers, pooling, and fully connected layers.
- Design a CNN model with:
  - Input layers matching frame dimensions.
  - Convolutional, pooling, and dense layers.
  - Output layer (softmax) for binary classification.
- Implement the model using TensorFlow/Keras.
- Train the model on a small subset of the dataset to debug.

Week 5: Model Training and Optimization
Regularization, BatchNorm, Dropout, and optimization algorithms.
Tasks:
- Train the CNN on the full dataset with basic hyperparameters.
- Implement BatchNorm to stabilize and accelerate training.
- Add Dropout layers to mitigate overfitting.
- Test optimization algorithms like SGD and Adam.
- Save model checkpoints and logs for analysis.

Week 6: Hyperparameter Tuning

Fine-tuning and performance optimization.

Tasks:

- Use tools like Keras Tuner to adjust:
  - Number of filters, kernel sizes, and activation functions.
  - Learning rates, batch sizes, and number of epochs.
- Compare models and select the best-performing configuration.
- Perform cross-validation to verify model robustness.

Week 7: Real-Time System Integration

Integrating models with external systems for deployment.

Tasks:

- Learn to use OpenCV for video frame capture.
- Integrate the trained model into a real-time detection script.
- Optimize the model for real-time inference (e.g., quantization, smaller batch sizes).
- Test the system on a live webcam or recorded video.

Week 8: Advanced Testing and Improvements

Error analysis and advanced CNN architectures.

Tasks:

- Test the system with a diverse set of videos to identify failure cases.
- Analyze errors and misclassifications using a confusion matrix.
- Enhance the model with advanced architectures like ResNet or EfficientNet.
- Re-train and re-integrate the improved model.

Week 9: Report Writing and Presentation

Technical writing and effective presentations.

Tasks:

- Write a structured report covering:
  - Problem definition.
  - Data preprocessing and augmentation.
  - Model design, training, and evaluation.
  - Real-time system integration.
- Include plots (e.g., accuracy/loss curves), diagrams, and test results.
- Prepare a concise presentation with visuals and a demo.

Week 10: Final Presentation and Submission

Project defense and evaluation.

Tasks:

- Deliver a 5-minute presentation summarizing key achievements.
- Demo the real-time detection system.
- Submit final deliverables: scripts, models, report, and presentation slides.
- Participate in Q&A and incorporate feedback.

6. Grading Rubric

7. Data Preprocessing: Quality of preprocessing scripts, data organization, and augmentation techniques (15%).
8. Model Design: Originality, architecture design, and effective implementation of CNNs (20%).
9. Model Training and Results: Use of optimization techniques, performance metrics, and improvements after tuning (25%).
10. System Integration: Functionality, speed, and accuracy of the real-time detection system (20%).
11. Documentation: Clarity, depth, and comprehensiveness of the project report (10%).
12. Presentation and Demo: Conciseness, clarity, and engagement during the final presentation and demo (10%).