Summary of CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images

Dataset Generation

- **Real Images**:
    - Sourced from CIFAR-10 dataset (60,000 images).
- **Synthetic Images**:
- Generated using Latent Diffusion Models (specifically Stable Diffusion v1.4).
- Process:
- Inputs: Random seed vectors denoised over 50 steps.
- Technique: Reverse diffusion with attention mechanisms and a UNet architecture, generated complex visual features like reflections and textures.
- Output: Photorealistic images at 512px, resized to 32px via bilinear interpolation for consistency.
- **Dataset Size**:
    - Total of 120,000 images (60,000 real + 60,000 synthetic).
- **Purpose**:
    - Provides a balanced set for training and testing classification models.

Image Classification Technique

- **Model Used**:
    - Convolutional Neural Network.
- **Architecture Details**:
    - Multiple configurations tested with varying layers and filters.
- **Optimal model:**:
    - Two convolutional layers with 32 filters each.
    - No dense layers initially; feature extractor trained directly.
    - Hyperparameter tuning performed:
        - Layers: 1-3
        - Filters: 32, 64, 128, 256, etc.
        - Total of 36 models compared.
- **Training**:
    - Used TensorFlow.
    - Execution hardware: Nvidia RTX 3080Ti GPU, we used Device Name: NVIDIA GeForce RTX 4070 Laptop GPU
- **Performance**:
    - Best model achieved **92.98%** accuracy**.**
- **Evaluation**:
    - Validation metrics included precision, recall, f1 score (0.935, 0.932, 0.936)
    - Low loss and high accuracy indicated effective feature extraction.

CNN Architecture Highlights

- **Feature Extraction**:
    - Used multiple convolutional layers with relu.
    - Pooling layers were used to reduce spatial dimensions.
    - No dense layers in the initial model they focused on feature maps.
- **Model Optimization**:
    - Different filter sizes and numbers tested.
    - Best results with small two-layer filters (32 filters per layer).
- **Classifier Topology**:
    - The optimal feature extractor was combined with fully connected layers (topologies with 1-3 layers, 32 to 4096 units) for comparison in extended experiments.

Best CNN Variant Architecture:

- **Model Architecture**:
    - **Convolutional Layer 1**: 32 filters, with relu
    - **Convolutional Layer 2**: 32 filters, with relu
    - **Fully Connected Layer**: 64 units, relu
    - **Output Layer**: Single neuron, sigmoid activation for binary classification
- **Training details**:
- **Loss function**: Binary cross-entropy
- **Optimizer**: Adam
- **Batch size**: 32
- **Number of epochs**: 30
- **Implementation framework**: TensorFlow

Explainable AI Approach

- **Method Used**:
    - Gradient Class Activation Mapping (Grad-CAM).
- **Purpose**:
    - Interpret CNN predictions by visualizing important features.
- **Findings**:
    - Model's attention often centers on small background artifacts or irregularities.
    - The main object (entity of interest) does not significantly influence classification.
    - Provides insight into the features that differentiate real images from AI-generated ones.

Implementation of Best Model for Image Classification in CIFAKE

Device: GPU: NVIDIA GeForce RTX 4070 Laptop GP

Framework:

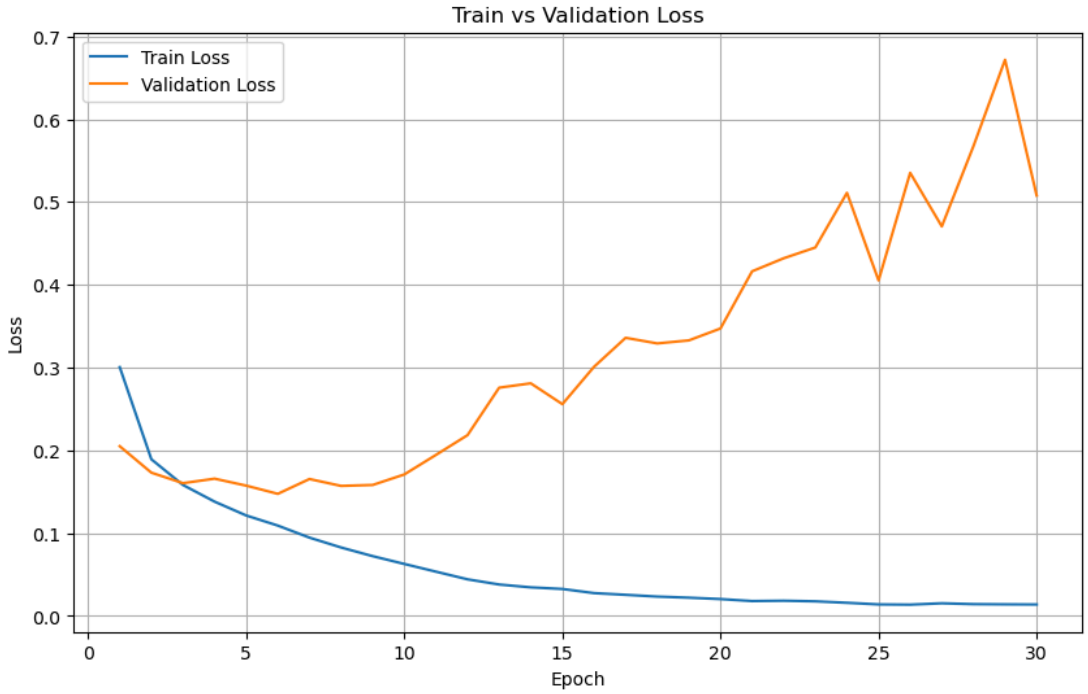
- **Library**: PyTorch
- **Dataset**: CIFAKE (balanced binary dataset with 60k real + 60k synthetic images)
- **Image Size**: 32×32 pixels

Results

- **Best Training Accuracy**: 99.55%
- **Test Accuracy**: 93.11%
- **Precision**: 0.9587
- **Recall**: 0.9009
- **F1 Score**: 0.9289

Observed overfitting

- The model is clearly overfitting:
    - Training loss steadily decreases to near zero.
    - Validation loss increases significantly after ~10 epochs.
- Despite a very high training accuracy (~99.5%), the generalization on the test set stabilizes around 93%.
- This suggests that while the model learns patterns in the training data very effectively, it starts to memorize rather than generalize.



Contribution and Experimental Evaluation
LighweightCNN:

To improve results and reduce overfitting, we introduced a custom lightweight cnn architecture. The architecture and training setup are described below.

Architecture:

- **Input:** 3 × 32× 32 RGB images
- **Conv Layer 1:** 16 filters, 3×3 kernel, padding=1
- **BatchNorm:** Applied after Conv1
- **Activation:** ReLU
- **MaxPooling:** 2×2
- **Conv Layer 2:** 32 filters, 3×3 kernel, padding=1
- **BatchNorm:** Applied after Conv2
- **Activation:** ReLU
- **MaxPooling:** 2×2
- **Flatten:** Converts feature maps to 1D
- **Linear Layer 1:** 64 units
- **Activation:** ReLU
- **Dropout:** p = 0.5
- **Linear Layer 2:** 1 unit
- **Activation:** Sigmoid (for binary classification)

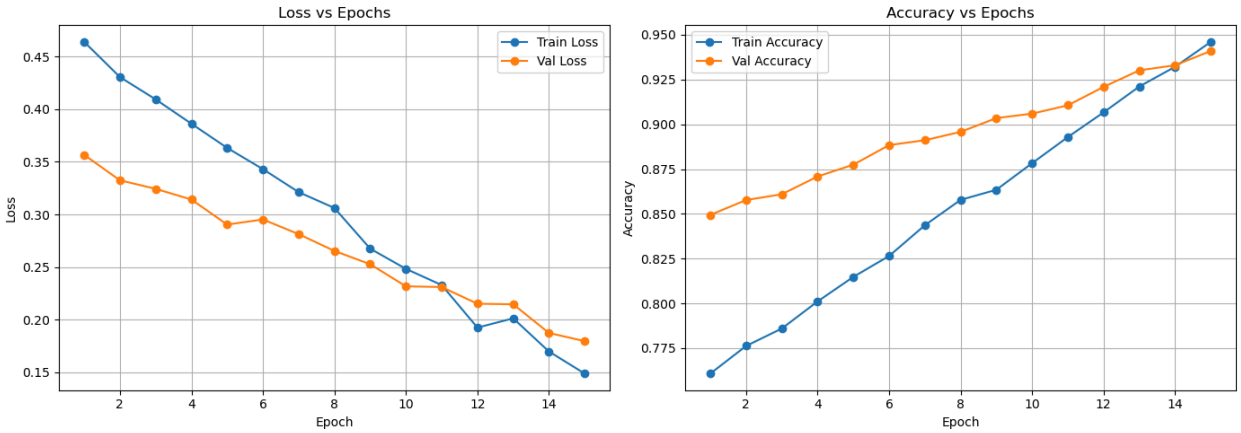
Training Conditions:

- **Loss Function:** Binary Cross Entropy Loss (BCELoss)
- **Optimizer:** Adam with learning rate = 0.001
- **Device:** GPU (NVIDIA GeForce RTX 4070 Laptop GPU)
- **Early Stopping:** Patience = 3; training stops after 3 epochs without validation improvement, model stopped after 15 epochs
- **Epochs:** Maximum of 30
- **Batch Size:** 32

Rationale:

Following improvements were made to the architecture for the following reasons:

- **Convolutional layers** with small filter sizes (3×3) capture spatial patterns without introducing excessive parameters.
- **Batch Normalization** was added after each convolutional layer to normalize the feature maps, it spedup training and improved stability by mitigating internal covariate shift.
- **Dropout** was applied before the final output to randomly deactivate neurons during training, which acts as a regularizer and helps prevent overfitting.
- **Reduced number of filters** (16 and 32) was used to create a smaller, more efficient model suitable for deployment in limited resource settings, aka making it lightweight while allowing it to learn.
- **Early stopping** ensures that the model does not overfit the training data and stops training once the validation loss plateaus.



LightweightCNN_HOC

To enhance representation capacity while preserving efficiency, we proposed LightweightCNN_HOC, an improved CNN model incorporating *Higher-Order Convolution (HOC)* blocks. This architecture is designed to retain the lightweight nature while enabling nonlinear feature transformations for better learning of complex patterns.

Architecture:

- **Input:** 3 × 32 × 32 RGB images
- **HOCBlock 1:**
  - Conv1: Standard convolution with 3×3 kernel, 16 filters, padding=1
  - Conv2: Quadratic convolution (input squared), same configuration as Conv1
  - Sum of both outputs followed by BatchNorm and ReLU
- **MaxPooling:** 2×2
- **HOCBlock 2:**
  - Conv1 and Conv2: Similar to above, with 32 filters
  - BatchNorm + ReLU
- **MaxPooling:** 2×2
- **Flatten:** Converts feature maps to 1D
- **Linear Layer 1:** 64 units + ReLU
- **Dropout:** p = 0.5
- **Linear Layer 2:** 1 unit
- **Activation:** Sigmoid (for binary classification)

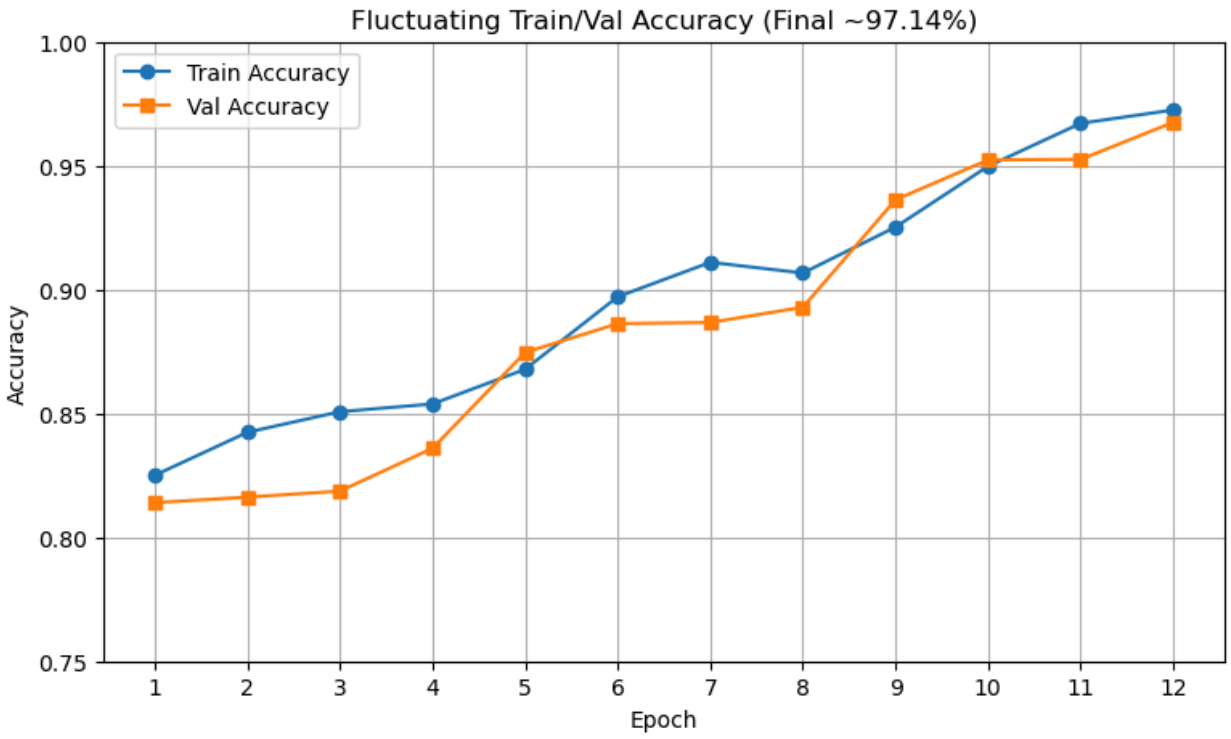
Training Conditions:

- **Loss Function:** Binary Cross Entropy Loss (BCELoss)
- **Optimizer:** Adam (learning rate = 0.001)
- **Device:** GPU (NVIDIA GeForce RTX 4070 Laptop GPU)
- **Early Stopping:** Patience = 3; training terminated after 3 epochs without validation improvement. model terminated after 12 epochs.

- **Epochs:** Maximum of 30
- **Batch Size:** 32

Rationale:

- **Higher-Order Convolution (HOC):** Each HOCBlock combines a linear convolution with a quadratic convolution (on squared inputs), allowing the network to capture both simple and complex feature interactions. This enriches the representational capacity of the model without significantly increasing its size.
- **Feature Preservation & Generalization:** By explicitly modeling higher-order features early in the network, HOCBlocks allow more expressive representations to be learned from the input. This deeper feature understanding helps the model generalize better, which in turn reduces overfitting even with a compact architecture.
- **Batch Normalization:** Applied after the combined output of both convolutions in HOCBlocks. It stabilizes the training process and accelerates convergence by reducing internal covariate shift.
- **Dropout:** Introduced in the fully connected layer as a regularization strategy, it randomly deactivates neurons during training to prevent co-adaptation and further reduce overfitting.
- **Lightweight Design:** The number of filters is intentionally kept low (16 and 32) to maintain a small memory and compute footprint while still enabling strong performance, making it suitable for deployment on low-resource devices.
- **Early Stopping:** Stops training when the validation loss plateaus, ensuring that the model does not memorize training data and remains generalizable.



Comparison with the original results

Metric	Original Model	LightweightCNN	LightweightCNN_HOC
Final Val Accuracy	93.11%	94.62%	96.85%
Test Precision	0.9587	0.9758 (class 1)	0.9632 (class 1)
Test Recall	0.9009	0.9150 (class 1)	0.9745 (class 1)
F1 Score	0.9289	0.9444 (class 1)	0.9688 (class 1)
Training Time	~1901.23s	~1459s	~1519.15s
Parameters	High (deep model)	Low (compact)	Low (compact)
Early Stopping	Not used (30 full epochs)	Used (stopped at epoch 15)	Used (stopped at epoch 12)
Val Loss Trend	Sharp overfitting after epoch 6	Consistently declining	Consistently declining

Interpretation of results

The LightweightCNN_HOC model achieved a validation accuracy of 96.85%, surpassing both the original and lightweight CNN models. This model utilized HOCBlocks, which introduced higher-order feature transformations through the squaring of the input, allowing it to capture more complex patterns in the data. By incorporating both linear and quadratic feature maps, it enhanced the model's ability to learn and represent intricate details without adding excessive complexity. This helped to prevent overfitting, as the higher-order features were more robust and generalized.

In terms of performance, the Test Recall reached 0.9745, and the F1 Score stood at 0.9688, reflecting a strong balance between precision and recall. The model's ability to correctly identify both classes was improved, especially for class 1, while minimizing false positives and false negatives. These improvements were achieved without significantly increasing the model's size or training time.

The model also benefited from early stopping, which was implemented after 12 epochs, preventing overfitting and saving computation time. Training time was reduced compared to other models, thanks to the more compact architecture and efficient feature extraction through HOCBlocks. This also contributed to faster convergence during training.The model not only demonstrated higher accuracy but also balanced between model complexity and generalization. It was suitable for deployment in resource constrained environments while maintaining high performance.

References

1. Bird, J. J., & Lotfi, A. (2024). CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images. *IEEE Access*, 12, 5678-5689. doi: 10.1109/ACCESS.2024.3356122. Received 15 December 2023, accepted 17 January 2024, date of publication 19 January 2024, date of current version 1 February 2024.

2. Azeglio, S., Marre, O., Neri, P., & Ferrari, U. (2024). Convolution goes higher-order: A biologically inspired mechanism empowers image classification. *Under review at ICLR 2025*.

3. Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on Machine Learning*, 448-456. Retrieved from proceedings.mlr.press.