Real and AI-Generated Images

Donna Qi
Department of ECE
University of Toronto
donna.qi@mail.utoronto.ca

Sophy Li
Department of ECE
University of Toronto
lsophy.li@mail.utoronto.ca

Ziyue Gong
Department of ECE
University of Toronto
joy.gong@mail.utoronto.ca

1 Introduction

The problem is to identify whether an image is a real-life photograph or AI-generated, using deep learning techniques. The motivation to solve this issue arises from the alarming surge in deepfake fraud in North America, which experienced a staggering 1740% increase [1]. Deepfake technology has enabled the creation of highly convincing fake images, videos, and audio, making it easier to manipulate media, spread misinformation, and commit identity fraud. This growing threat motivates us to address the problem with deep learning, as AI-generated images may contain patterns that are challenging for traditional algorithms to detect. Problems with a natural hierarchy of features—such as images, where edges form shapes and shapes form objects—are ideal for deep networks. These networks can learn feature hierarchies across multiple layers, enabling more accurate detection of AI-generated content.

2 Preliminaries and Problem Formulation

The ultimate goal of this project is to develop a robust deep learning model that can accurately distinguish between authentic and AI-generated images. This capability is essential for mitigating risks associated with deepfake content, including identity fraud, misinformation, and media manipulation. By training a model to detect subtle differences in image features, we seek to contribute a tool that can support the authenticity of visual media and reduce the impact of deceptive AI-generated content.

3 Solution via Deep Learning

Datasets

We will primarily use the CIFAKE dataset **[2]** from Kaggle, which includes a mix of real and AI-generated synthetic images. The entire dataset contains 100k training images and 20k test images. For the REAL class, images are sourced from the well-known CIFAR-10 dataset by Krizhevsky & Hinton, which contains natural images across 10 object categories. The FAKE images, on the other hand, are generated to closely resemble CIFAR-10 images using Stable Diffusion version 1.4. The method reverses the process of Gaussian noise, to generate synthetic images from noise. Mathematically speaking, a noisy image x_t is generated from the original x_0 by:

$$x_t = \sqrt{\bar{a}_t}x_0 + \sqrt{1 - \bar{a}_t}\varepsilon$$

where noise is ε, and the adjustment according to the time step t is $\bar{a}$. Reverse diffusion reverses the process of 50 noising steps, which from x_{50}, will yield x_0, the synthetic image.

We will not be performing validation on the CIFAKE dataset unless they are not performing well. In that case, validation will be used to further tune hyperparameters.

As a stretch, we want to see how the models can identify fake images generated using a different generator. Therefore, a secondary dataset, which is a Shoes dataset **[3]** will be used. These images are generated via the Midjourney generator.

Data Preprocessing

Preprocessing the dataset will allow us to normalize images from various sources, so input to the model is consistent. The images will be resized, normalized and converted to a common shape 32x32x3.

Data splitting

The full training set across all CIFAR classes is split FAKE:50k and REAL:50k. The full test set is split FAKE:10k, REAL:10k. The project will use a subset of the train and test data to reduce computation time. The training set and test set will be randomly shuffled, and 25k will be used as a training subset (FAKE: 12.5k, REAL: 12.5k). Similarly, the test set will be shuffled, and 5k will be used as a testing subset (FAKE: 2.5k, REAL: 2.5k). Afterwards, the train data will be loaded with a batch size of 32, and the test data will be loaded with a batch size of 64.

Model Overview

We plan to try 2 types of models, one is a simple CNN built from scratch, the other are pre-trained ResNet models to see how they compare when identifying AI images.

Model1: simple CNN

This model is built from three interchanging Conv2D and max pooling layers, followed by a flattening, dense, dropout, and dense layer [4]. The convolution layers extracts important features into a feature map that is then reduced via max pooling to control overfitting, while retaining dominant features. We will use a flatten layer to transform the multidimensional output from the previous convolution layers to a 1D vector, then feed the result into a dense RELU layer. The dense layer combines extracted features from previous layers as the final output for prediction. We apply dropout, then the dense sigmoid layer is used for classification of the image as AI or real. The CNN architecture is shown in the below figure:

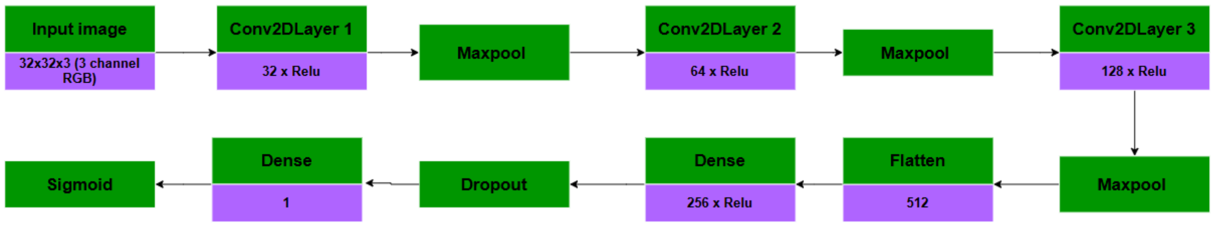


Figure 1: Flow diagram of simple CNN model

Model2: ResNet

Resnet is a family of deep convolutional neural networks to ease the training of deep networks. This project will leverage 5 pre-trained ResNet models offered by torchvision: ResNet-18 v1, ResNet-50 v1, ResNet-50 v2, ResNet-101 v1, and ResNet-101 to see how well they compare to the CNN model in classifying images as real or fake. Each Resnet model will aim to train to a target accuracy of 95% and early stopping of 10 epochs.

A table comparing the features of the different ResNet models is below:

Table 1: Comparison table of ResNet18, ResNet 50, and ResNet 101 [5][6][7]

Feature	**ResNet-18**	**ResNet-50**	**ResNet-101**
Depth	18 layers	50 layers	101 layers
Parameters	11.7M	25.6M	44.5M
Block type	Basic block	Bottleneck	Bottleneck

Size of weights	~45MB	~98MB	~171MB
Performance	Faster training	Balanced	High capacity
Interference Time	Fast	Moderate	Slowest
Applications	Lightweight, edge devices	General- purpose tasks	Large-scale tasks, fine-grained tasks

V2 ResNet models differ from V1 in that it does pre-activation rather than post-activation. This means, batch normalization is before the activation function, which is before the convolutional layers. This is expected to mitigate vanishing gradients more effectively and produce better results.

4 Implementation

Design

The design involves building models to classify real versus AI-generated images using the CIFAKE dataset.

Modules & libraries used

- PyTorch: For model implementation and training.
- Torchvision: For data transformations and pre-trained ResNet models.
- Scikit-learn: For evaluation metrics like confusion matrices, ROC-AUC, and log loss.
- Matplotlib/Seaborn: For plotting performance metrics and visualizations.
- kagglehub: For downloading datasets

Define test, train, and evaluation functions

1.1 Define Test
- Set no gradients, loop through batched train subset, compute average risk (loss_function(outputs, labels)) and accuracy (fraction of correctly predicted labels) over the train batches

1.2 Define Train
- The optimizer will use Adam optimizer with learning rate of 0.0001.
- Loss: binary cross-entropy loss to measure the difference between the predicted probabilities and the actual labels.
- Baseline: an untrained CNN model using the test subset. Both risk and accuracy is calculated to compare with trained models.
- Training loop: for num_epochs, get average (over the batches) risk and accuracy for test and train sets
- Loop until MAX_EPOCH=10, or test accuracy above 95%

Pseudo-code for the training loop:

```
def train(model, num_epochs, loader, optimizer, loss_fn):
  for epoch in range(num_epochs):
    for images, labels in loader:
      images, labels = images.to(device), labels.to(device).float().view(-1, 1)
      optimizer.zero_grad()
      outputs = model(images)
      loss = loss_fn(outputs, labels)
      loss.backward()
      optimizer.step()
```

1.3 Define Evaluation

- Predictions are converted to binary: fake or real
- The primary metrics we will look at are train risk, test risk and test accuracy. Further metrics provided by sklearn include confusion matrix, AUC, Log Loss, Matthew's Correlation Coefficient, Balanced Accuracy and Cohen's Kappa. Details on these performance metrics can be found in the Appendix.

Models

2.1 Simple CNN

The architecture of the simple CNN is shown in Table 2.

Table 2: Architecture of simple CNN model

Layer Type	Details	Input Shape	Output Shape
Input Layer	Shape (32, 32, 3) - RGB Channels	(32, 32, 3)	(32, 32, 3)
Conv2D Layer 1	32 filters, kernel size (3x3), ReLU	(32, 32, 3)	(30, 30, 32)
MaxPooling2D 1	Pooling with 2x2 window	(30, 30, 32)	(15, 15, 32)
Conv2D Layer 2	64 filters, kernel size (3x3), ReLU	(15, 15, 32)	(13, 13, 64)
MaxPooling2D 2	Pooling with 2x2 window	(13, 13, 64)	(6, 6, 64)
Conv2D Layer 3	128 filters, kernel size (3x3), ReLU	(6, 6, 64)	(4, 4, 128)
MaxPooling2D 3	Pooling with 2x2 window	(4, 4, 128)	(2, 2, 128)
Flatten Layer	Converts 2D feature map to 1D vector	(2, 2, 128)	512 1D vector
Dense Layer 1	Fully connected layer with 256 neurons	512 1D vector	256 1D vector
Dropout Layer	Randomly sets outputs to zero during training	256 1D vector	256 1D vector
Dense Layer 2	Final fully connected layer (output)	256 1D vector	1

2.2 ResNet

ResNet models 18, 50 and 101 were chosen to represent a spectrum of different depths in a neural network. Furthermore, different versions of each model (v1 and v2) are analyzed to see how pre-activation and post-activation differs in model accuracy. Details are shown in Table 2 below. Note, ResNet18 does not have a v2 model provided by TorchVision, so only the v1 model is used.

Table 3: Architecture of various ResNet models [5][6][7]

Feature	**18 v1**	**50 v1**	**50 v2**	**101 v1**	**101 v2**
Architecture	18 layers, 2-layer residual blocks	50 layers, bottleneck blocks (3 layers/block)	Same, but with pre-activation in bottleneck blocks	101 layers, bottleneck blocks (3 layers/block)	Same, but with pre-activation in bottleneck blocks

Algorithms

- Binary Cross-Entropy Loss: Since we are determining if the input image is real or AI-generated, this is a binary classification problem. Binary cross-entropy loss is used to measure the difference between the predicted probabilities and the actual labels.
- Adam Optimizer: Most effective optimizer for model parameters with adaptive learning rates.
- Early Stopping: To prevent overfitting during training.

5 Numerical Experiments

Dataset Config

- **Training/Testing Split**: Balanced (Note) sets of 25k training and 5k testing samples were created, normalized, and augmented for training.
- **Additional Dataset**: Additional testing was conducted using the “shoe” dataset, resized to match the CIFAKE input size. This is to see how the models generalize for AI datasets created using a different generator.

Note: For the train and test dataset subset, we considered a random subset, or balanced subset (25k train, 5k test). Random subset is easier to obtain with `.random_split,` but it leads to a biased untrained model. Balanced subset ensures equal number of images across the 10 categories (airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks), i.e. for training, 25k/10 for each category; for test, 5k/10 for each category.

Model Config

CNN: 3 layers of convolution (each: Conv2d + ReLu + Max pooling), 1 flatten layer, 2 fully connected layers (activated respectively by ReLu, Sigmoid). Detailed parameters mentioned above in Part3, Model Overview.

ResNet: 5 different ResNet models: ResNet-18, ResNet-50 and ResNet-101 and their pre-trained weights.

Training Config

- Batch Size: 32 for training and 64 for testing.
- Loss function: BCELoss
- Optimizer & Learning Rate: Default learning rates from PyTorch optimizers Adam.
- Early stop: Trained for up to 10 epochs with a target accuracy of 95%.

Experiments and Results

Table 4: Train risk, test risk and test accuracy for the different models using CIFAKE dataset

Model	**Epochs**	**Train Risk**	**Test Risk**	**Test Accuracy**
Untrained CNN	NA	NA	0.6934	0.4982
CNN	10 (max)	0.1013	0.2583	0.9114
Resnet 18 v1	10 (max)	0.0535	0.1809	0.9365
Resnet 50 v1	10 (max)	0.1477	0.1870	0.9337
Resnet 50 v2	10 (max)	0.0698	0.1278	0.9569
Resnet 101 v1	10 (max)	0.4140	0.9894	0.8208
Resnet 101 v2	10 (max)	0.0625	0.3388	0.8881

Table 5: Test risk and test accuracy using additional Shoe dataset

Model	**Test Risk**	**Test Accuracy**
Untrained CNN	0.6981	0.3549
CNN	2.8354	0.5426
Resnet 18 v1	1.7939	0.5447
Resnet 50 v1	1.4016	0.5465
Resnet 50 v2	2.2100	0.5555
Resnet 101 v1	5.9554	0.3114
Resnet 101 v2	4.5177	0.4665

Note: see Appendix for more metrics

Learning Curves

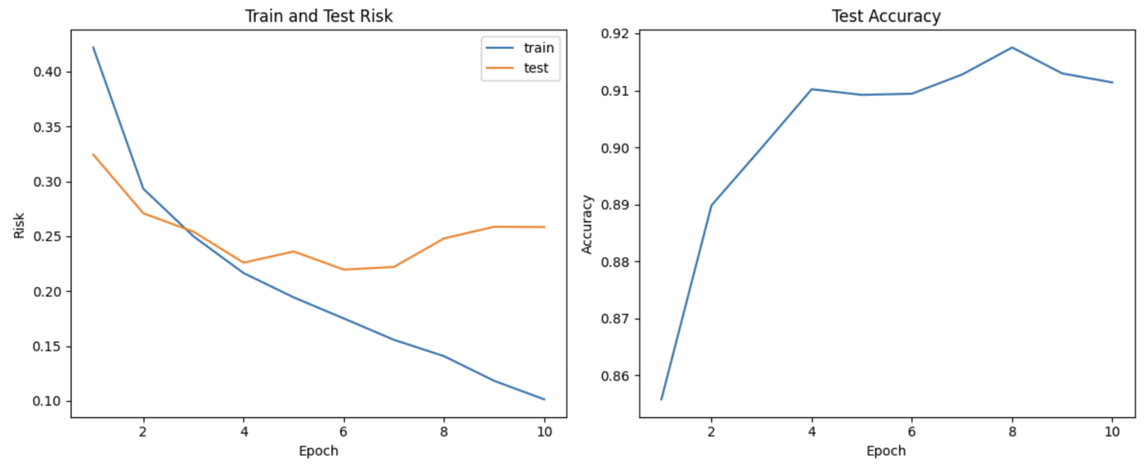


Figure 2: Train risk, test risk and test accuracy for simple CNN model

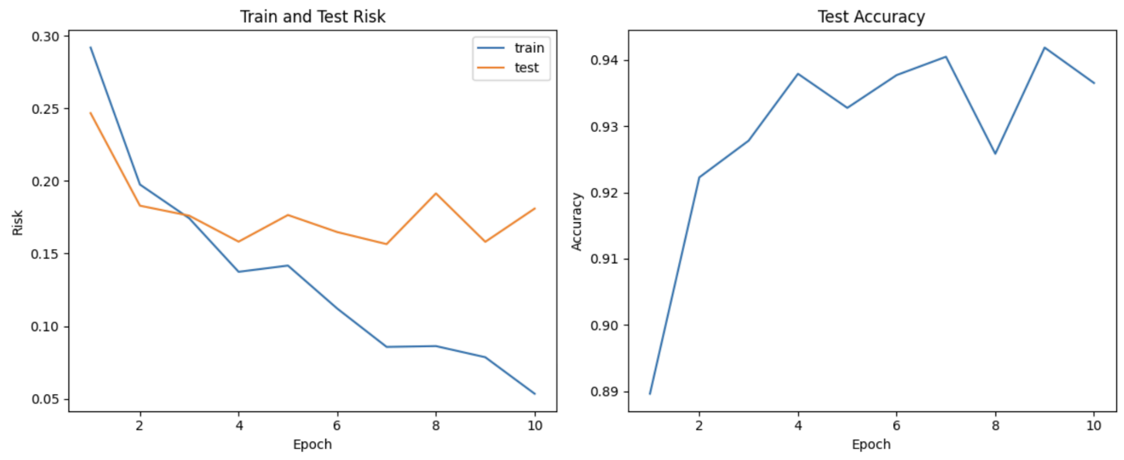


Figure 3: Train risk, test risk and test accuracy for simple ResNet18 v1 model

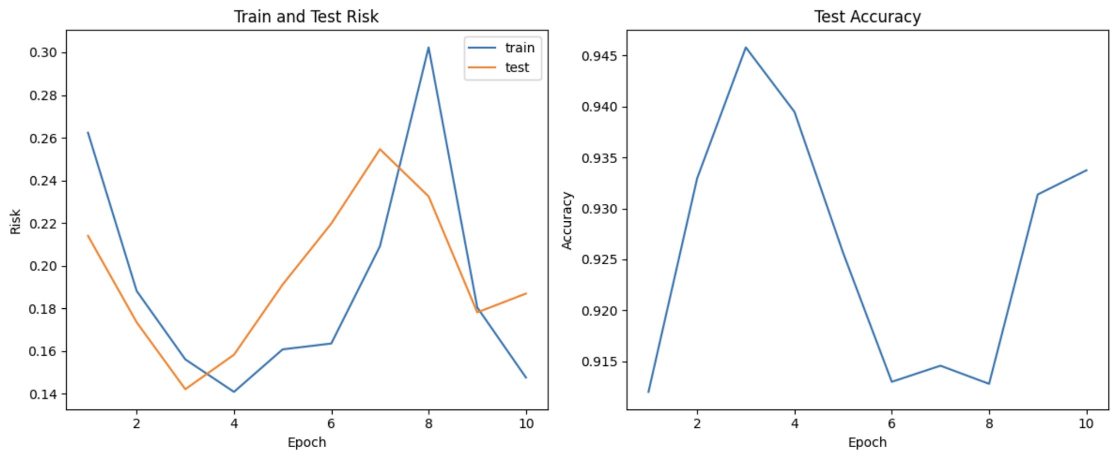


Figure 4: Train risk, test risk and test accuracy for ResNet50 v1model

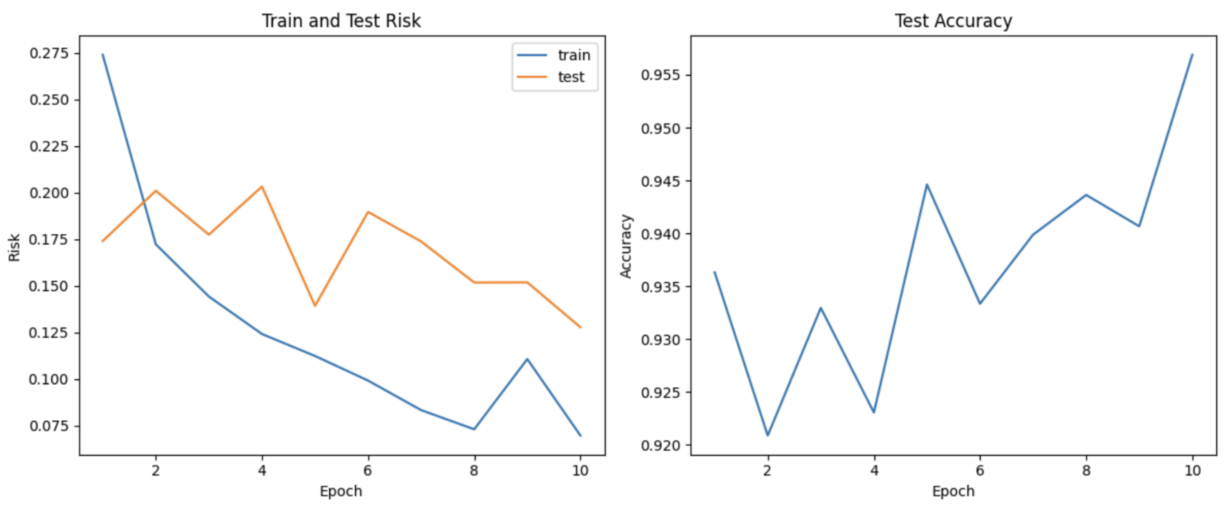


Figure 5: Train risk, test risk and test accuracy for ResNet50 v2 model

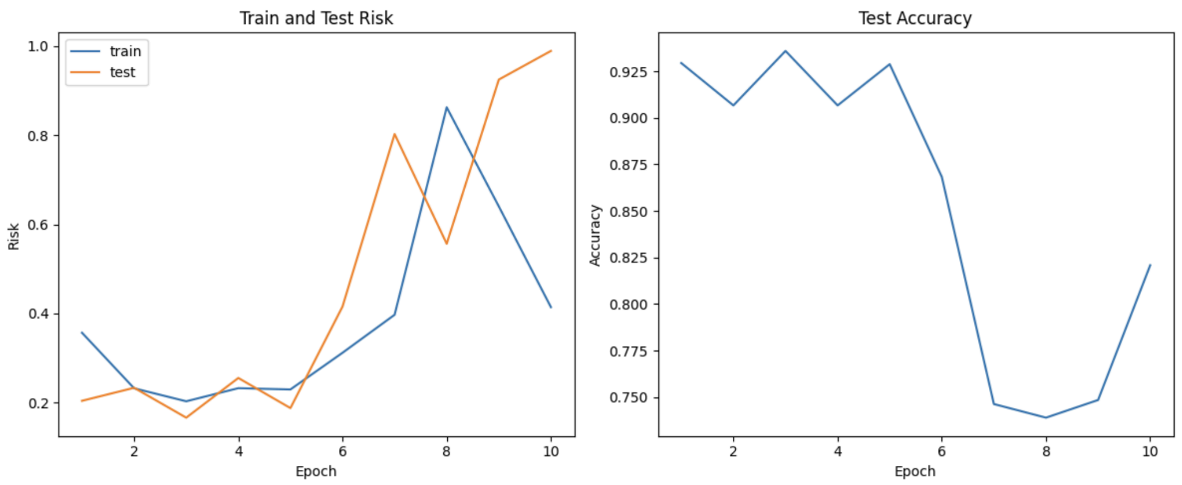


Figure 6: Train risk, test risk and test accuracy for ResNet100 v2model

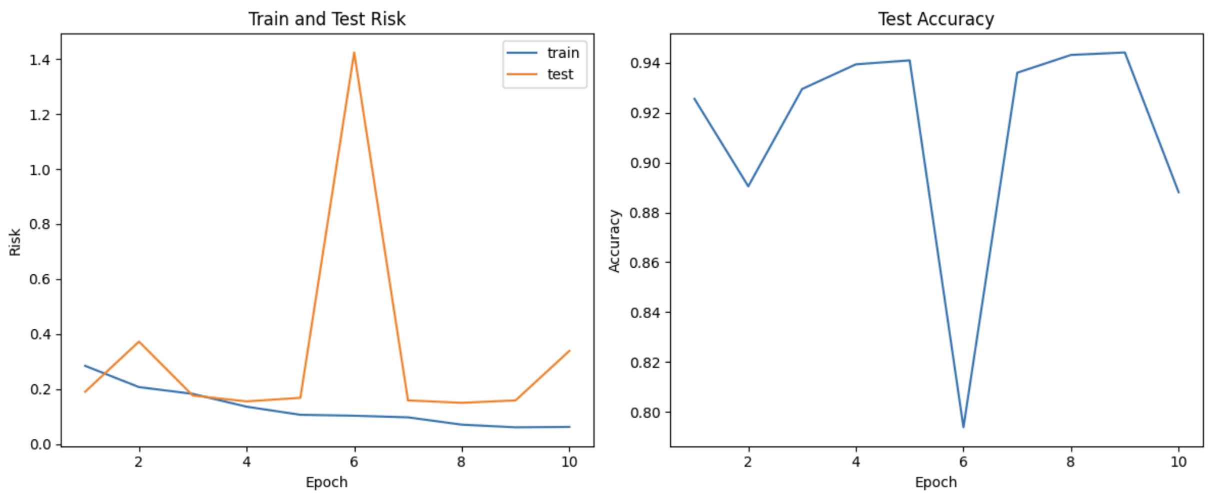


Figure 7: Train risk, test risk and test accuracy for ResNet100 v2 model

6 Answer Research Questions

The primary research question addressed was: *Can simple CNN architectures achieve comparable performance to pre-trained deep networks for CIFAKE-like datasets?*

- Simple CNNs, when trained effectively, achieved an accuracy of ~91% (Table 4) with a slightly lower AUC-ROC compared to ResNet models.
- Pre-trained ResNet architectures offered superior generalization and robustness, particularly for smaller datasets or transfer learning tasks.

Do these models perform well given datasets built using a different generator?

- There is low validation accuracy using the Shoes dataset. The most accurate model is ResNet50 v2 achieving 55.6% test accuracy (Table 5).
- Test accuracy for the trained CNN model is 54.26%, and mean test accuracy for the ResNet models is 48.49% (Table 5). Low accuracy indicates the models were unable to generalize for a new synthetic dataset built using a different generator.

7 Conclusions

Findings

- Best model: ResNet 50-v2 achieved the best test accuracy of 95.69% and the lowest test risk of 0.1278, making it the most effective model overall (Table 4).
- CNN
  - Risk plot in Figure 2 indicates overfitting starting around epoch 6, as the test risk increases and test accuracy fluctuates.
  - Early stopping and choosing a proper target accuracy to stop at are essential to balance performance and prevent overfitting.
- ResNet **v2 vs. v1 Stability:**
  - Overall, v2 exhibits less frequent, and/or less steep oscillations ("triangles") in the risk plot compared to their v1 counterparts. This stability is attributed to differences in residual block design and training mechanisms
  - ResNet v1**:** Applies batch normalization (BN) and activation (ReLU) after convolutional layers (post-activation).
  - ResNet v2**:** Applies BN and ReLU **before** convolutional layers (pre-activation).
  - Pre-activation normalizes outputs first and then adds them to the residual connection, allowing for more stable gradient flow (avoiding vanishing or exploding gradients) and reducing oscillations, particularly in deeper architectures.
- ResNet **Depth Comparison:**
  - Depth 18: Trains the fastest but is limited in representational power, resulting in lower performance.
  - Depth 101: Exhibits the most instability, likely due to its increased complexity. This can be seen in Figure 6, where the test accuracy dips around epoch 7. More layers can be harder to converge, resulting in a lower train and test score compared to the shallower models.
  - Depth 50: Strikes the best balance between complexity and performance, offering the highest accuracy with stable training behavior.
- ResNet vs CNN
  - **Performance Comparison:** CNN achieved 91.71% test accuracy, which is significantly lower than all ResNet models, particularly ResNet 50-v2 (95.69%)

- ○ **Limitations of CNN:** The shallow nature of CNN architectures restricts their ability to capture complex patterns, especially when compared to ResNet models that incorporate residual connections and deeper architectures for enhanced representational power.

Future Improvements

- As can be seen from Table 5, neither CNN nor ResNet generalize well across different generators. The test accuracy is much lower compared to test accuracy using CIFAKE. Mentioned in Section 3 Solution via Deep Learning, the train and test dataset CIFAKE is generated with Stable Diffusion v1.4 [2]; while the additional dataset shoe is generated with Midjourney [3]; Both models performed well for CIFAKE dataset but not generalize to show dataset. Future work could involve exploring other architectures that generalize better across different generators used for generating AI images; a potential solution is detection by rich and poor texture contrast [10].

References

[1] "Sumsub Expert Roundtable: The Top KYC Trends Coming in 2024", Dec 27, 2023 Sumsub Blog
[2] J. J. Bird and A. Lotfi, "CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images," in IEEE Access, vol. 12, pp. 15642-15650, 2024, doi: 10.1109/ACCESS.2024.3356122.
keywords: {Artificial intelligence;Visualization;Data models;Image recognition;Computational modeling;Synthetic data;Image classification;AI-generated images;generative AI;image classification;latent diffusion},
Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.
[3] S. Kakar, "Shoes Dataset: Real and AI-Generated Images," Kaggle Datasets. [Online]. Available: https://www.kaggle.com/datasets/sunnykakar/shoes-dataset-real-and-ai-generated-images?select=ai-midjourney. [Accessed: Dec. 6, 2024].
[4] Simple CNN on CIFAKE
https://www.kaggle.com/code/muntasirfahimniloy/simple-cnn-on-cifake-90-acc
[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," arXiv:1512.03385, Dec. 2015. [Online]. Available: https://arxiv.org/abs/1512.03385. [Accessed: Dec. 6, 2024].
[6] K. He, X. Zhang, and S. Ren, "Identity Mappings in Deep Residual Networks," arXiv:1603.05027, Mar. 2016. [Online]. Available: https://arxiv.org/abs/1603.05027. [Accessed: Dec. 6, 2024].
[7] "ResNet Implementation in PyTorch," Neurohive, [Online]. Available: https://neurohive.io/en/popular-networks/resnet/. [Accessed: Dec. 6, 2024].
[8] "Performance Metrics in Machine Learning," Delft University of Technology, [Online]. Available: https://3d.bk.tudelft.nl/courses/geo5017/handouts/09_notes_PerformanceMetrics.pdf. [Accessed: Dec. 6, 2024].
[9] "What Are Performance Metrics in Machine Learning?" Robots.net, [Online]. Available: https://robots.net/fintech/what-are-performance-metrics-in-machine-learning/. [Accessed: Dec. 6, 2024].
[10] “Detection of AI generated images using rich and poor texture contrast”, Hriday Keswani, [Online]. Available: https://medium.com/@hridaykeswani/detection-of-ai-generated-images-using-rich-and-poor-texture-contrast-fc2024e3e716
[Accessed: Dec. 13, 2024]

Supplementary

Performance metrics used to evaluate model performance [8][9]

Performance Metric	Description	Pass condition
Train risk	Measures the average loss (error) on the training dataset. Lower values indicate better model fit.	<0.5 (low train risk)
Test Risk	Measures the average loss on the test dataset. It reflects generalization performance.	<0.5 (low test risk)
Test Accuracy	Fraction of correct predictions over the total predictions on the test set.	>90% (high accuracy)
Confusion matrix	Tabular summary of correct and incorrect predictions across all classes (True Positives, False Negatives, etc.).	High diagonal dominance
AUC-ROC	Area under the Receiver Operating Characteristic curve. Higher values indicate better classification performance.	Close to 1
Log loss	Logarithmic loss that penalizes incorrect predictions with a confidence score. Lower is better.	Minimized (< threshold)
Matthews Correlation Coefficient	Correlation between true and predicted classifications; values range from -1 (worst) to 1 (best).	Close to 1
Balanced accuracy	Average recall obtained on all classes, particularly useful for imbalanced datasets.	High (e.g., >80%)
Cohen's capa	Measures agreement between observed and predicted classifications, adjusted for random chance. Values range from 0 to 1.	Close to 1

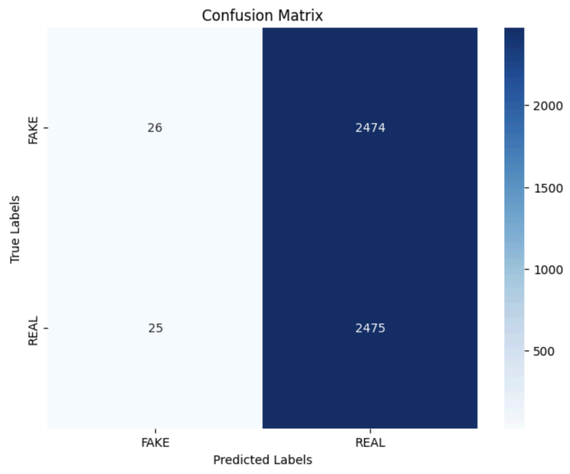
Model results using additional performance metrics

Model	AUC-ROC	Log loss	MCC	Balanced accuracy	CC	F1-score
Untrained CNN	0.4563	0.6935	0.0002	0.5002	0.0004	0.34
Trained CNN	0.9713	0.2605	0.8224	0.9104	0.8208	0.91
Resnet 18 v1	0.9821	0.1825	0.8716	0.9358	0.8716	0.94
Resnet 50 v1	0.9809	0.1795	0.8689	0.9344	0.8688	0.93

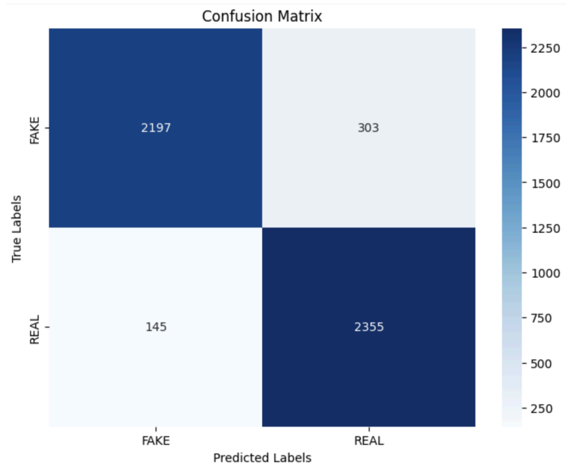
Resnet 50 v2	0.9902	0.1290	0.9128	0.9564	0.9128	0.96
Resnet 101 v1	0.8787	0.7678	0.6431	0.8202	0.6404	0.82
Resnet 101 v2	0.9845	0.3426	0.7912	0.8868	0.7736	0.89

Confusion matrix

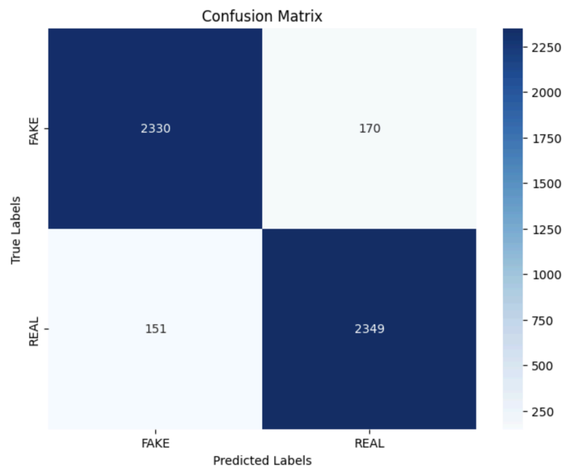
Untrained



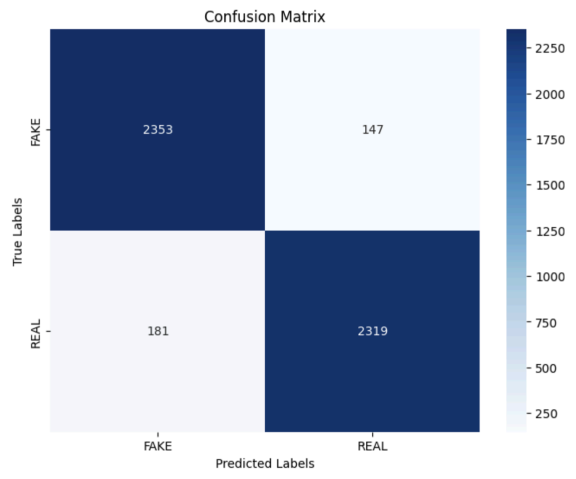
CNN

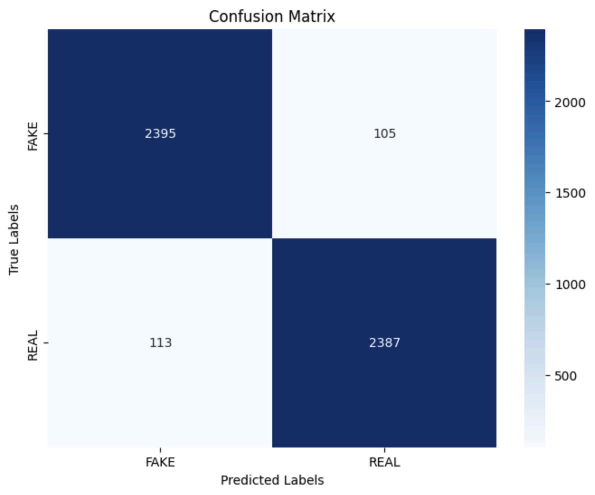


ResNet 18-v1

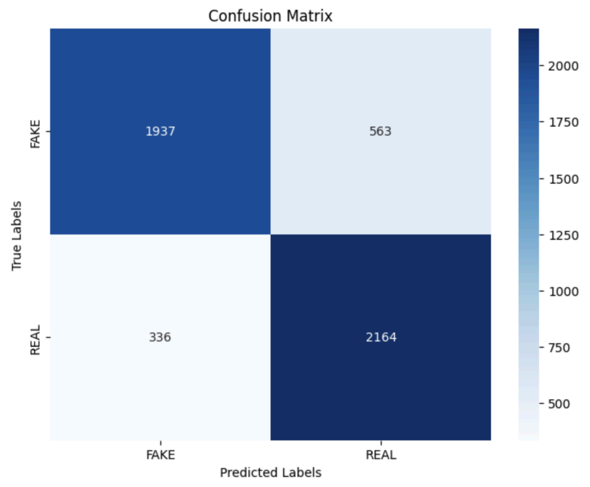


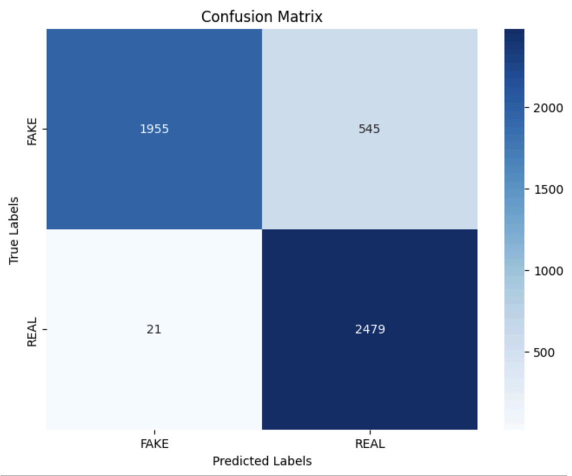
ResNet 50-v1, 50-v2





ResNet 101-v1, 101-v2


Confusion Matrix
FAKE
REAL
True Labels
Predicted Labels


Confusion Matrix
FAKE
REAL
True Labels
Predicted Labels