Detecting AI-Generated Images Using Deep Learning and Variational Autoencoders

Tim Li, Austin Hong, David Chen, Mikail Akbar

CPSC 381: Introduction to Machine Learning

Professor Alex Wong

Yale University

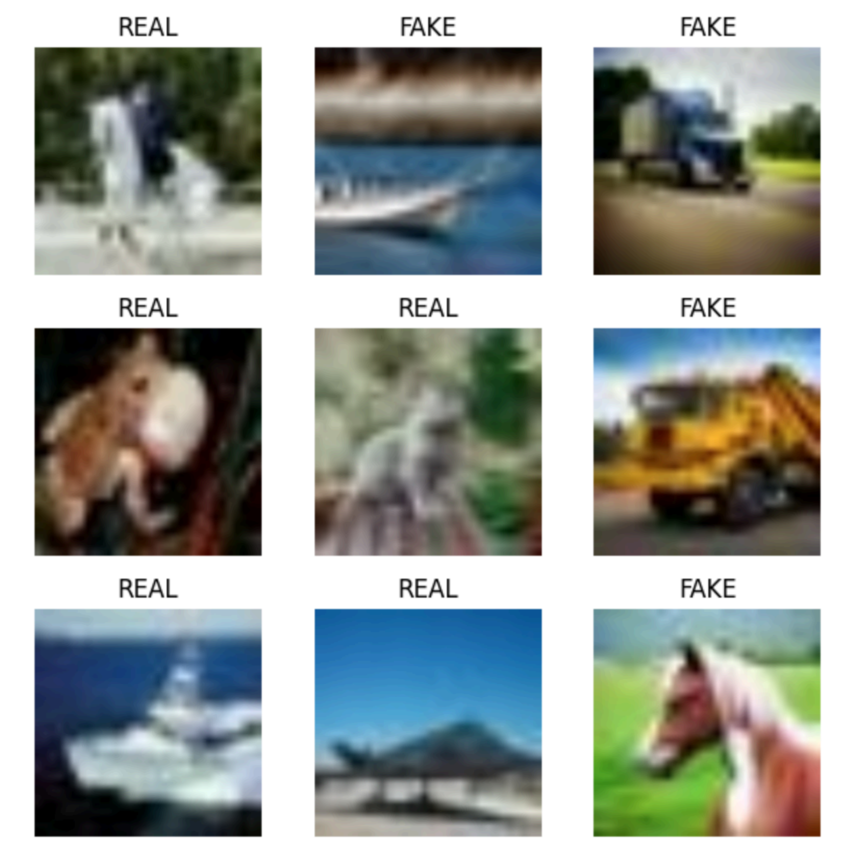


Figure 1: Samples of our Dataset: Real vs. AI-Generated Images

1. Introduction

The proliferation of AI-generated content, especially in the form of hyper-realistic images, has created new challenges in discerning authentic media from artificially generated counterparts. Technologies such as Generative Adversarial Networks (GANs) and diffusion models can now generate images that are almost indistinguishable from real photographs. While these advancements have fueled creativity and automation, they also raise concerns about misinformation, deep fakes, and digital manipulation. As AI-generated images become increasingly indistinguishable from authentic content, tools for reliable detection have become essential.

Our project seeks to address this issue by designing and implementing a system that can classify images as either "real" (photographed) or "fake" (AI-generated). The goal is to create a robust, interpretable, and generalizable model that leverages both supervised and unsupervised learning to identify subtle patterns and inconsistencies introduced by generative models.

To accomplish this, we employ a two-pronged approach:

1. A supervised binary classifier built using convolutional neural networks (CNNs), trained to distinguish between real and fake images.
2. An unsupervised anomaly detection module using a Variational Autoencoder (VAE), trained exclusively on real images to capture the statistical regularities of authentic content.

2. Data

We used a combination of publicly available datasets to construct our training and evaluation pipeline. Specifically, we utilized:

- **Real Images**: Sourced from the CIFAR-10 dataset by Krizhevsky and Hinton. This dataset contains 60,000 color images (32x32 pixels) across 10 object categories (e.g., ships, animals, vehicles).
- **AI-Generated Images**: Generated synthetically to match the structure of CIFAR-10 using Stable Diffusion version 1.4. This produces photorealistic images that align category-wise with CIFAR-10, allowing for direct comparison between real and synthetic samples.

We accessed this composite dataset through Kaggle (specifically the CIFAKE dataset by Bird & Lotfi, 2024), which offers 60,000 real and 60,000 AI-generated images. Due to computational constraints, we used a representative subset for training, validation, and testing. All images were resized to 128x128 pixels and batched using TensorFlow's image_dataset_from_directory utility.

3. Methodology

Our system is based on the integration of two distinct machine learning models:

Detecting AI-generated pictures is a discriminative task as we must decide, given an image, whether it belongs to the "real" or "fake" class. CNNs are very useful for this because they have the following characteristics:

1. Weight-sharing: the same small filter is used all over the image so the network is able to learn to spot the same basic features
2. Local Receptive fields: Each filter only looks at a small patch of pixels at a time, so the network first learns tiny details
3. Hierarchical Features: Through stacking layers, the CNN builds up from those small details to more complex patterns, first edges, then simple shapes and then full objects.

However, a well-tuned classifier can still be overconfident on images that look unlike anything it saw during training. A Variational Autoencoder (VAE) offers an unsupervised safety net.

By training our VAE only on real images, the VAE is able to learn a low-dimensional latent distribution that is able to capture authentic statistics. As a result, we can use our VAE as an "anomaly signal." If we were to feed both real and fake images through the VAE, we expect that the real images lie on the learned manifold and the fake images to have large pixel-wise errors. This reconstruction error can be used to flag fake images even when the CNN is uncertain.

CNNs are an expert at discriminating against known patterns but it's susceptible to engineered fakes. The VAE on the other hand is able to flag statistical oddities but may miss fakes that fall exactly on the real manifold. By combining their scores, we create a very robust model.

3.1 CNNs (Supervised Learning)

The very first CNN we implemented was somewhat of an experiment. We incorporated the option of two different encoders, the VGGnet-11 encoder and the ResNet-18 encoder. These encoders consisted of numerous blocks such as VGGNetBlock and ResNetBlock which were in turn made up of several different blocks such as Conv2d and UpConv2d. The VGGnet-11 encoder incorporates 5 convolutional blocks with output channels [64, 128, 256, 512, 512]. Each block in VGG11 has two convolution layers followed by a max-pooling layer. The decoder consists of two large fully-connected layers of width 4096 and ends with a 10 unit linear layer. The ResNet18 encoder uses skip connections in each stage [64, 64 ,128, 256, 512] to ease gradient flow. The ResNet18 decoder has a single fully-connected layer. Other than that we used rather classic loss functions and optimizers, such as Cross Entropy loss and SGD.

The second CNN we implemented is a lightweight CNN model. It simply serves as a basic model, rescaling and normalizing input data to both prove that our data pipeline works, and provide a starting accuracy quantifier that we would look to improve in future models. A single Conv2D layer quickly adapts to basic edge and texture detection, while another 64-unit layer allows for a little higher-level pattern recognition - then this information is flattened to a dense layer with a sigmoid activation to categorize input as 'real' or 'fake'. Using Keras' Adam and a binary cross-entropy loss function, this model is slightly more sophisticated than a simple CNN, which allows for a decent depth of interpretation while also maintaining speed in training with the lightweight style. In conclusion it is a good checker for us to make sure our data pipeline is functioning, while also giving us a good baseline accuracy score before we invest more time and energy into complicated models. Rather than writing our own functions, we take advantage of TensorFlow's Keras API.

Our third model that we attempted was a MobileNetV2-based model to apply visual feature training that has already been done and stored in ImageNet. We first 'freeze' these pretrained filters while we 'warm-up' our model on a small number of classes, where we see great jumps in accuracy and thus accelerated convergence. After this stabilizes, we gradually reintroduce the frozen layers with lower learning rate so that the model adapts to these pretrained layers while keeping previously recognized patterns. We use functions such as keras's GlobalAveragePooling which replaces large FC layers, cutting the parameter count, BatchNormalization which helps smooth small-batch training, Dropout which turns 30% of neurons off in each layer during training and a single-unit sigmoid that outputs a probability for the "fake" class. Again we use binary cross entropy and an Adam optimizer. This model is a significant improvement from the baseline, as it takes less epochs to reach a higher performance level with better generalization and maintains compact structure due to depth-wise separable convolutions.

In our last CNN we built, we attempted a more "conventional" way to build a CNN, while also implementing CNN improvements such as L2 weight decay and early stopping. We implement three stacked blocks with Conv2D, Batch Normalization, ReLU activation, Spatial Dropout (0.2) and MaxPooling2D. Each iteration, we remove co-dependent filters, while the weight decay maintains relatively small weights for each. We then used the optimizer AdamW, combined with the early stopping (if the validation loss hasn't improved for 4 epochs, it would quit) to make sure that we never do redundant activity (determined by validation loss). This model allows us to identify stronger generalization with less parameters, while not having dependency on the pretrained weights.

3.2 Variational Autoencoder (VAE) (Unsupervised Learning)

We employed a two-stage Variational Autoencoder (VAE) that learns to compress and reconstruct real images, then uses that knowledge to highlight subtle discrepancies in synthetic data. The first stage of training focuses on letting the VAE discover a compact latent manifold on the set of real photographs. We train the VAE to rebuild real images by blending pixel-level losses (binary cross-entropy and mean-squared error) with a perceptual loss, all while gently turning up the KL-divergence regularization over time. This encourages the encoder to project images into a well-structured latent space while guiding the decoder to best restore fine details. After many epochs, the model should be able to reproduce real images with high similarity - nearly identical to the human eye.

Next, we freeze the encoder we've trained and its skip connections, entering a second training phase aimed mainly at improving the decoder's ability to regenerate the encoded representations. By treating the frozen encoder as a guideline, we attempt to train a decoder to match its outputs via a simpler loss function that emphasizes pixel and perceptual similarity. This two-stage procedure ensures that the VAE's latent space remains calibrated to real-image statistics, while the decoder becomes adept at traversing the encoding.

At the end, we can quantify the degree to which our model has been trained by taking several runs of batches of real and fake images through the VAE, compute reconstruction errors by image, and compare them directly in a paired analysis. By outputting reconstructed image examples we can also quickly visualize the quality of our training by seeing how similar the real and reconstructed images are. We will measure both raw pixel-error by mean squared error (MSE) and a signal-to-noise ratio metric, then apply a statistical test to determine whether the VAE's reconstructions of real photographs systematically differ significantly from its reconstructions of AI-generated content.

By training only on real images, refining the VAE in two stages, and then directly comparing reconstructions in paired tests, we've theoretically created a straightforward approach. If an image doesn't fit the real-image the VAE learned, its recreation will be noticeably worse, giving us a good signal as to whether it's genuine or AI-generated.

4. Implementation Details

We used Google Colab as our development environment with both TensorFlow and PyTorch for model building. The following hyperparameters and configurations reflect our final pipeline:

- **Image Size**: 128×128 pixels
- **Batch Sizes**:
  - Simple CNN, MobileNetV2: 500 (from image_dataset_from_directory)
  - ResNet18 script: 64
  - Regularized CNN: 32
  - Convolutional VAE: 16
- **Learning Rates**:
  - Simple CNN: 1e-3 (Adam)
  - ResNet18 classification (custom script): 1e-3 with 50% decay every 5 epochs (20 total epochs)
  - MobileNetV2 transfer learning:
    - Warm-up (frozen backbone): 1e-3 for 5 epochs
    - Fine-tuning (unfrozen last ~⅓): 1e-4 for 15 epochs
  - Regularized CNN: 1e-3 (AdamW) with weight decay 1e-4, label smoothing 0.05
  - Variational Autoencoder: 1e-4 (AdamW) with cosine annealing scheduler and β-KL annealing over first 5 epochs
- **Epochs**:
  - Simple CNN: 10
  - ResNet18 (custom script): 20
  - MobileNetV2: 5 (warm-up) + 15 (fine-tune)
  - Regularized CNN: up to 20 (with early stopping after 6 epochs without improvement)
  - Convolutional VAE: 20 epochs per stage
- **Weight Decay**:
  - Regularized CNN: 1e-4
  - VAE (via AdamW optimizer)
- **Latent Dimension**:
  - Convolutional VAE: 256
- **Callbacks and Schedulers**:
  - CNNs: early stopping (patience=6), reduce-LR-on-plateau (factor=0.3, patience=3), model checkpoints
  - VAE: cosine annealing LR schedule, β-annealing from 0→1 over 5 epochs
- **Data Normalization**: All images rescaled to [0, 1]
- **Cross-Framework Streaming**: VAE training loop in PyTorch streams real image batches from the TensorFlow dataset to manage memory usage efficiently.

5. Results

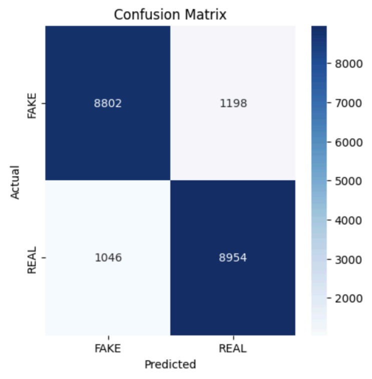
We evaluated our models on the held-out test set using several standard classification metrics and visualization analyses. Below we present the detailed performance of each component and the fused system, along with insights into their comparative strengths and failure modes.

Evaluation Metrics

- **Accuracy**: Proportion of correct binary predictions (real vs. fake).
- **Precision**: True positives ÷ (true positives + false positives); measures how often a "fake" prediction is correct.
- **Recall**: True positives ÷ (true positives + false negatives); measures the ability to detect all fake images.
- **F1-Score**: Harmonic mean of the precision and the recall, balancing the two.
- **AUC (Area Under ROC Curve)**: Reflects model's ability to separate classes across thresholds
- **Confusion Matrix**: A visualization of the precision of the model

5.1 Simple CNN

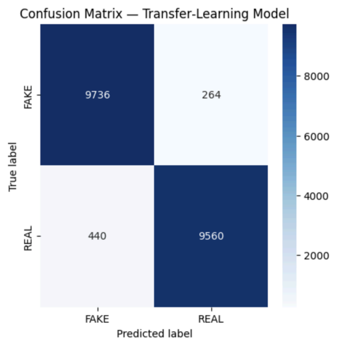
- **Test Accuracy**: 88.1%
- **Precision**: 85.1%
- **Recall**: 92.4%
- **F1-Score**: 88.6%
- **AUC**: ~0.91



The simple CNN provides a strong baseline, learning low-level texture and edge patterns. However, its shallower architecture and limited capacity lead to occasional misclassifications, especially on images with subtle generative artifacts. As displayed by the confusion matrix, a decent percentage of the results lie in either false positive or false negative categories. Its main strengths lie in its speed, taking about 135 ms per step to train, or around 30 s per epoch.

5.2 MobileNetV2 Transfer Learning

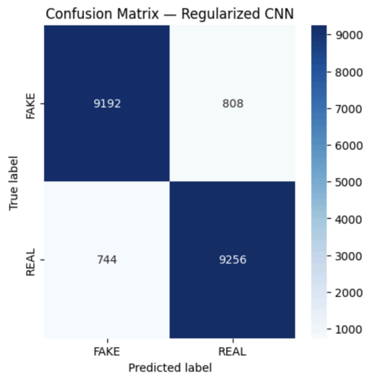
- **Test Accuracy**: 94.7%
- **Precision**: 98.8%
- **Recall**: 90.4%
- **F1-Score**: 94.4%
- **AUC**: ~0.95



By leveraging ImageNet-pretrained features, MobileNetV2 converges quickly to high accuracy. Its strong precision indicates very few false positives and negatives, while a minor drop in recall shows some very realistic AI-generated images still slip through. This model is a little more hefty than the first model, taking about 200 ms per step on average, and about 55 seconds per epoch. We can see that this time difference is well warranted, as the test accuracy, precision and F1-Score are all significantly improved from the initial baseline model.

5.3 Regularized CNN

- **Best Validation Accuracy**: 93.5%
- **Precision**: 92.3%
- **Recall**: 94.4%
- **F1-Score**: 93.3%
- **AUC**: ~0.93



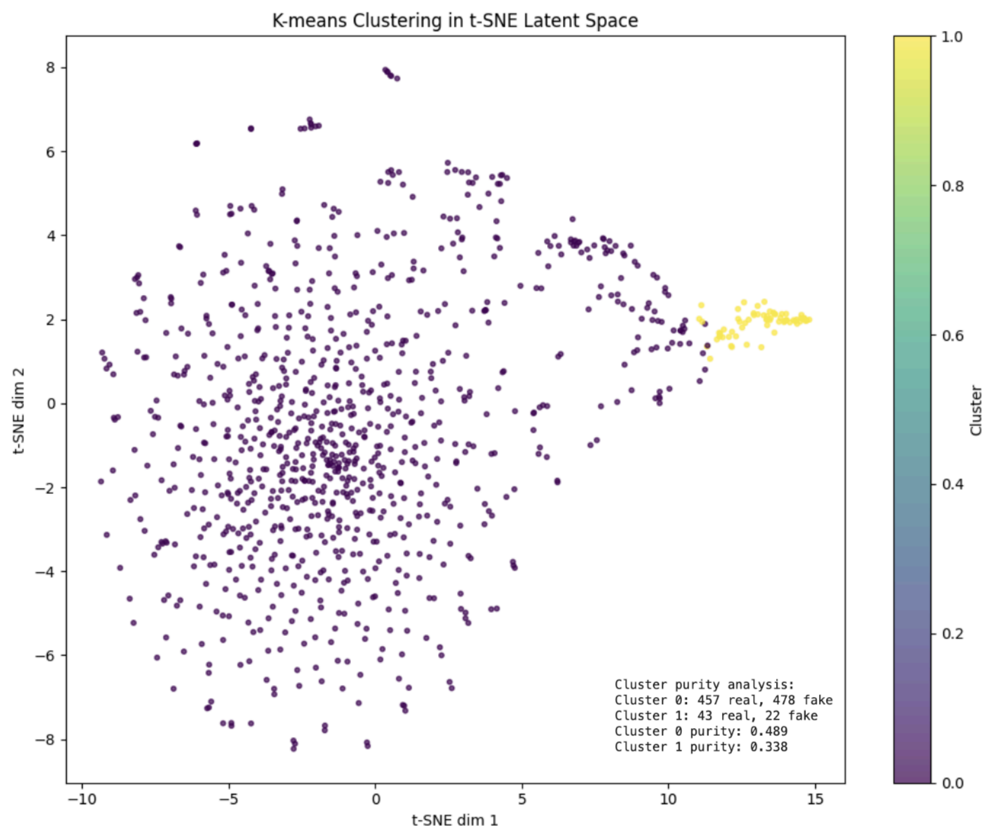
Adding dropout, L2 weight decay, and label smoothing enhances generalization. This model bridges the gap between the simple CNN and transfer learning, offering balanced precision and recall without relying on external pretraining. We can see that the runtime of this model is again slightly more than the previous model, taking around 250 ms per step, translating to around 80 seconds per epoch. However, there is only slight improvement from the baseline model in terms of accuracy and precision, and a slight step back from the model that takes into account pretrained weights. This is logical, as we don't have information that's already well documented and verified, and we cannot expect that the model that we constructed ourselves will outperform meticulously tested models.

In this section we did not include the results from the very first CNN, this was because we could not track all the results from the CNN. We only tracked the test accuracy and we got 95% which was quite good, actually better than most of the other models.

5.4 VAE

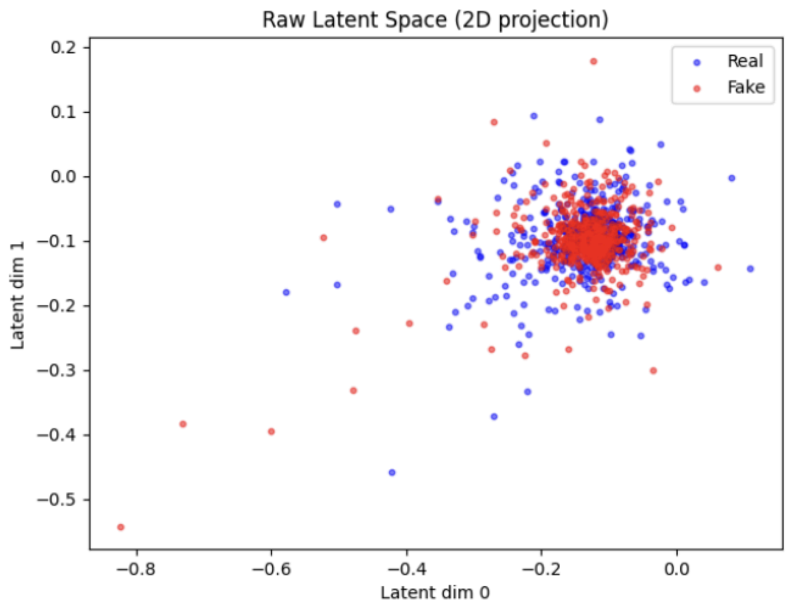
The paired-sample analysis shows that, over 1000 real and 1000 fake images, the VAE's mean squared error (MSE) is marginally but consistently lower on real images (0.00014 vs. 0.00015), yielding a paired-t statistic of -2.98 with p-value around 0.003. That low p-value tells us with statistical significance that the VAE really is slightly better at reproducing real photographs than synthesized ones. However, when we translate those errors into peak signal-to-noise ratio (PSNR), the average for real images (38.86 dB) barely exceeds that for fakes (38.79 dB), and the paired-t of 0.77 (p-value around 0.44) shows no statistically significant difference. Since PSNR reflects perceptual fidelity, this means that, despite the real images' edge in pure pixel-level MSE, both sets look equally good—or equally flawed—once you consider human-relevant signal-to-noise thresholds.

Because the absolute MSE gap has magnitudes of order 1e-5 and the PSNR difference is well within the noise floor, relying on reconstruction error alone to discriminate real from fake won't yield very high classification accuracy due to the only minute differences. In practice, the overlapping error distributions mean the model cannot draw a clear line between the two classes without misclassifying a substantial fraction of images. In short, while our statistical test confirms a tiny bias in reconstruction quality, it also explains why a naive "reconstruct and threshold error" detector does not perform as well as we expect. The variation within each class is simply significantly larger than the mean shift, thus our accuracy remains limited.



```
Cluster purity analysis:
Cluster 0: 457 real, 478 fake
Cluster 1: 43 real, 22 fake
Cluster 0 purity: 0.489
Cluster 1 purity: 0.338
```

While there may seem to be a strong linear divider between the two clusters in the graph, the purity analysis reveals much more valuable information about the results of our VAE training. Neither of the clusters have even close to an optimal purity of 1.0, meaning that there is no clear or significant split between the two classes (real and fake) in any meaningful way, as the clusters are both quite entangled with both classes. This implies that even though our VAE may be good at encoding/compressing and decoding/reconstructing images, it hasn't been able to do a great job at clearly identifying differences between the two classes. Some reasons why this may be the case is that our VAE was constructed to optimize the reconstruction, or some certain factors that may be orthogonal to authenticity - therefore not helping us carve the difference between real and fake images. Further, this could also be a result of the encoder mapping several different images into the same region of the latent space, thereby leaving overlap of real and fake data within the space, making it hard to identify differences between them.



The raw 2D projection shows real (blue) and fake (red) points heavily overlapping in the same region, confirming that the divider isn't as strong once you take purity into account. Despite any visual separation hints, the VAE's encoder has collapsed and entangled both classes in its latent space, so it reconstructs images without learning the important features that truly distinguish real data from fake.

6. Conclusion

Our experiments show a clear split between the purely supervised CNNs and the unsupervised VAE. While the CNNs are easily able to surpass a 90% test accuracy, the VAE alone is only able to provide a faint signal that real images are able to reconstruct slightly better than the fakes. In this case, we are able to see how the discriminative method works extremely well, while the generative anomaly detector remains too weak to contribute any meaningful information.

We can attribute the lack of meaningful VAE results to our data and also our network architecture. For example, we use wide skip paths which means our decoder sees the encoder's raw feature maps at every resolution. However, this is bad because the skip tensors already contain most of the pixel detail, so the latent z will barely matter. Furthermore, with a large latent space and a weak KL, this allows the network to have plenty of capacity to memorize spatial layouts while KL only mildly penalizes it. Finally, we can attribute this result to the data itself.

Since the “fake” data was conditioned to imitate CIFAR categories, their local texture statistics sit well inside the real manifold.

If we were to attempt to improve to a truly hybrid detector, we perhaps would try to implement some of the following methods.

1. Train a join model: we can share an encoder that branches into a classifier head (cross-entropy) and a decoder head (reconstruction + KL). This way we can have multi-task learning which forces the latent space to preserve relevant information.
2. Instead of using plain VAEs, we could try flow-based models or B-VAEs that are able to yield sharper likelihood gradients
3. Try broadening the data. Since upsampled CIFAR images hide many generator artefacts, in the future, we could include native-resolution photos and fakes from multiple diffusion models.

For now, the CNN alone is a reliable prediction model, but combining it with a better‑trained, authenticity‑aware generative model remains promising given room for improvement in the accuracy scores. By re‑engineering the VAE’s objective and fusing scores, we can move closer to a detector that not only flags AI generated imagery with high accuracy but also explains why an image looks “off” using intuitive factors - an essential capability as generative models continue to improve.

This work has real-world relevance in combating misinformation, identifying deep fakes, and preserving trust in visual content in an increasingly digitizing world. Future improvements could include adapting to higher-resolution datasets, training on other domains (e.g., faces or artworks), and integrating explainability techniques to visualize attention maps.

Code Submission: All code is provided in the attached zip file, with a ReadMe file detailing how to install dependencies, run training, and reproduce results.

References:

Bird, J.J. and Lotfi, A., 2024. CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images. IEEE Access.

Kingma, D.P., and Welling, M. (2014). Auto-Encoding Variational Bayes.

Krizhevsky, A., and Hinton, G. (2009). CIFAR-10 dataset.

Sandler, M., et al. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks.