Applications of Machine Learning Methods: Real vs. AI-Generated Image Classification

0. Introduction

The rapid advancement of Artificial Intelligence (AI) has led to powerful image generation models, such as Stable Diffusion, Midjourney, and DALL-E, capable of creating highly realistic synthetic images. While these technologies offer creative potential, they also pose significant challenges related to the spread of misinformation and the erosion of trust in digital media. Distinguishing between authentic photographs and AI-generated fakes has become increasingly important. This project tackles this challenge by applying machine learning techniques to classify images based on their authenticity.

The primary objective of this project is to develop and evaluate machine learning models capable of accurately classifying images as either authentic photographs or synthetically generated by AI. We aim to compare the performance of different approaches to understand their effectiveness in this task. This project utilizes a specific dataset containing images from the aforementioned AI generators and various real image sources (Pexels, Unsplash, WikiArt). We will explore a baseline model and move on to more advanced models as we go.

This report details the characteristics of the selected dataset, the methodology employed for data preprocessing and model development, presents the evaluation results comparing the models, and discusses the findings, limitations, and potential real-world applicability of this work.

1. Dataset Selection & Exploration

https://www.kaggle.com/datasets/tristanzhang32/ai-generated-images-vs-real-images

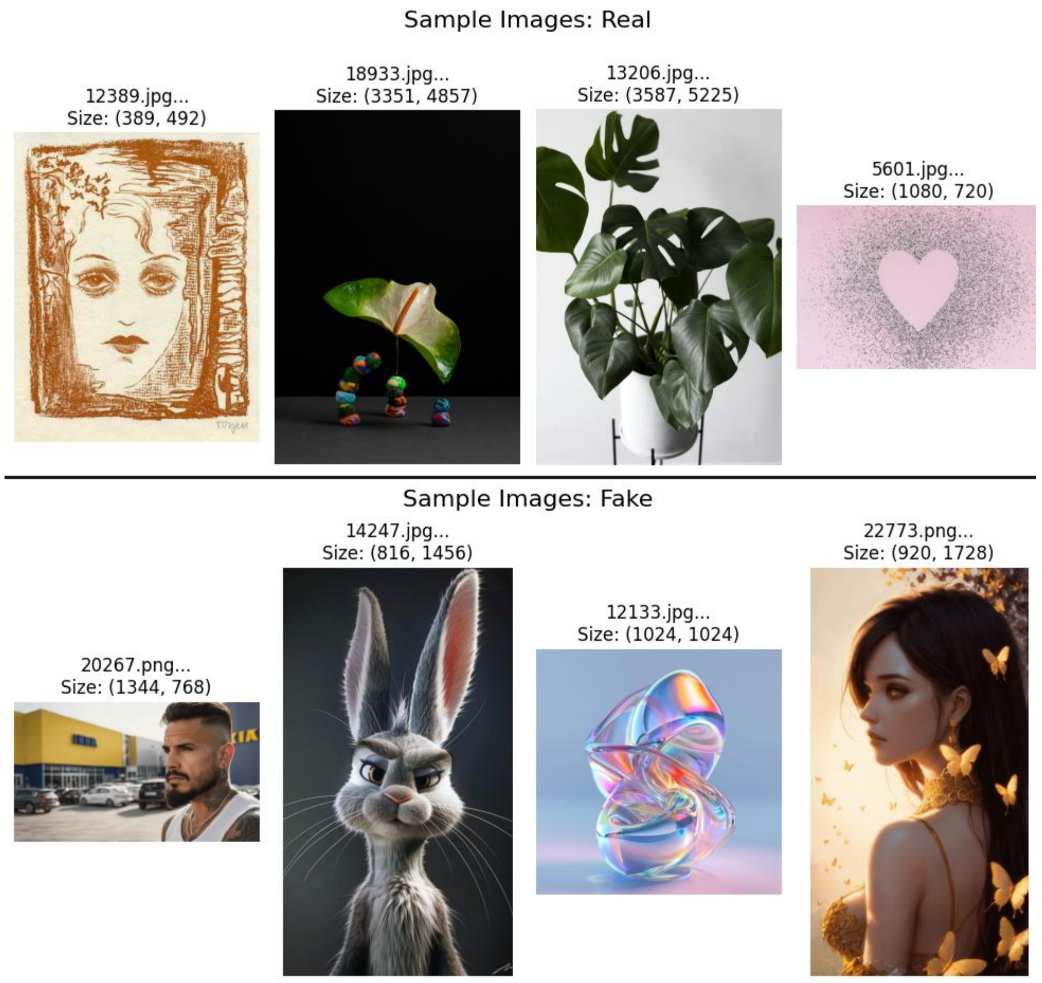
Why we chose this dataset: We chose the "Real vs. Fake Face Detection" dataset because it directly addresses our objective of distinguishing AI-generated images from real ones, a critical issue given the rise of sophisticated AI image synthesis. This dataset is particularly suitable due to its substantial size (60,000 images), which provides a robust foundation for training deep learning models. Furthermore, it includes images from multiple, highly relevant AI generators (Stable Diffusion, Midjourney, DALL-E) and diverse sources of real images (Pexels, Unsplash, WikiArt). This variety is crucial for developing a potentially more generalizable detection model compared to one trained on fewer sources.

1.1. Exploring the dataset

Fake Images (30,000 total): 10,000 from Stable Diffusion, 10,000 from MidJourney, 10,000 from DALL-E.

Real Images (30,000 total): 22,500 from Pexels/Unsplash, 7,500 from WikiArt. The key information for each image is the image itself and its label ('real' or 'fake'). The labels seem to come from the subfolders the images are in.

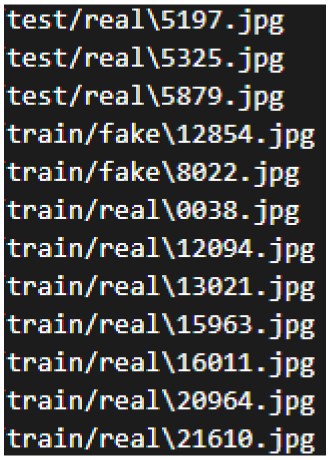
Visual Inspection: Sample images were visually inspected from both 'real' and 'fake' categories. This initial check provided a qualitative sense of the image styles and content diversity within each class and revealed a diverse range of subjects and styles within both classes, with no immediately obvious, universal visual artifacts distinguishing all fake images at first glance other than text being slightly distorted on the first fake image.



Generated in the EDA.ipynb file under the 'Show Sample Images' section.

1.2. Exploratory Data Analysis (EDA)

Corrupt Data: A data integrity check was performed using a script that attempted to open and verify each image file in the dataset using the Pillow library's verify() and load() methods. This process identified 12 files that could not be properly read, indicating corruption. Errors encountered included "image file is truncated" and "broken data stream when reading image file". These corrupted files were found across both training and testing sets (7 in train/real, 2 in train/fake, 3 in test/real) and their specific paths have been logged.



```
test/real\5197.jpg
test/real\5325.jpg
test/real\5879.jpg
train/fake\12854.jpg
train/fake\8022.jpg
train/real\0038.jpg
train/real\12094.jpg
train/real\13021.jpg
train/real\15963.jpg
train/real\16011.jpg
train/real\20964.jpg
train/real\21610.jpg
```

Generated by the bad_image_detector.py file to a .txt file.

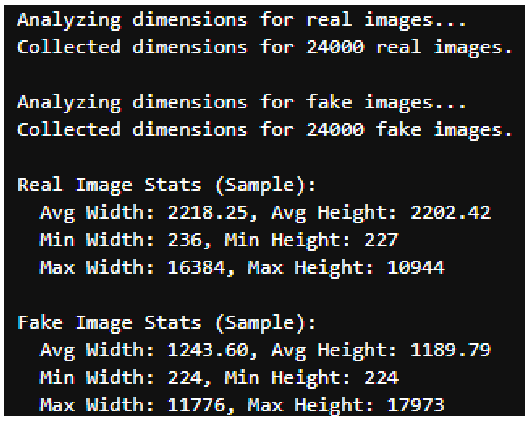
Structure: The dataset is already split into 80% for training and 20% for testing. The images are organized into 'fake' and 'real' subfolders.

Large Image Handling: During dimension analysis, several 'real' images exceeded Pillow's default maximum pixel limit (89,478,485 pixels), with the largest containing approximately 161 million pixels. DecompressionBombWarnings were triggered. Given the dataset source is considered reliable and sufficient system RAM is available, the Pillow MAX_IMAGE_PIXELS limit was increased to 200,000,000 to allow processing of these legitimate large images during analysis and subsequent steps.

Image Dimensions: Analysis of all 24,000 'real' and 24,000 'fake' images in the training set revealed significant variation in resolutions.

- Real Images: Average dimensions were approximately 2218x2202 pixels, with a range from 236x227 (Min Width/Height) up to 16384x10944 (Max Width/Height).

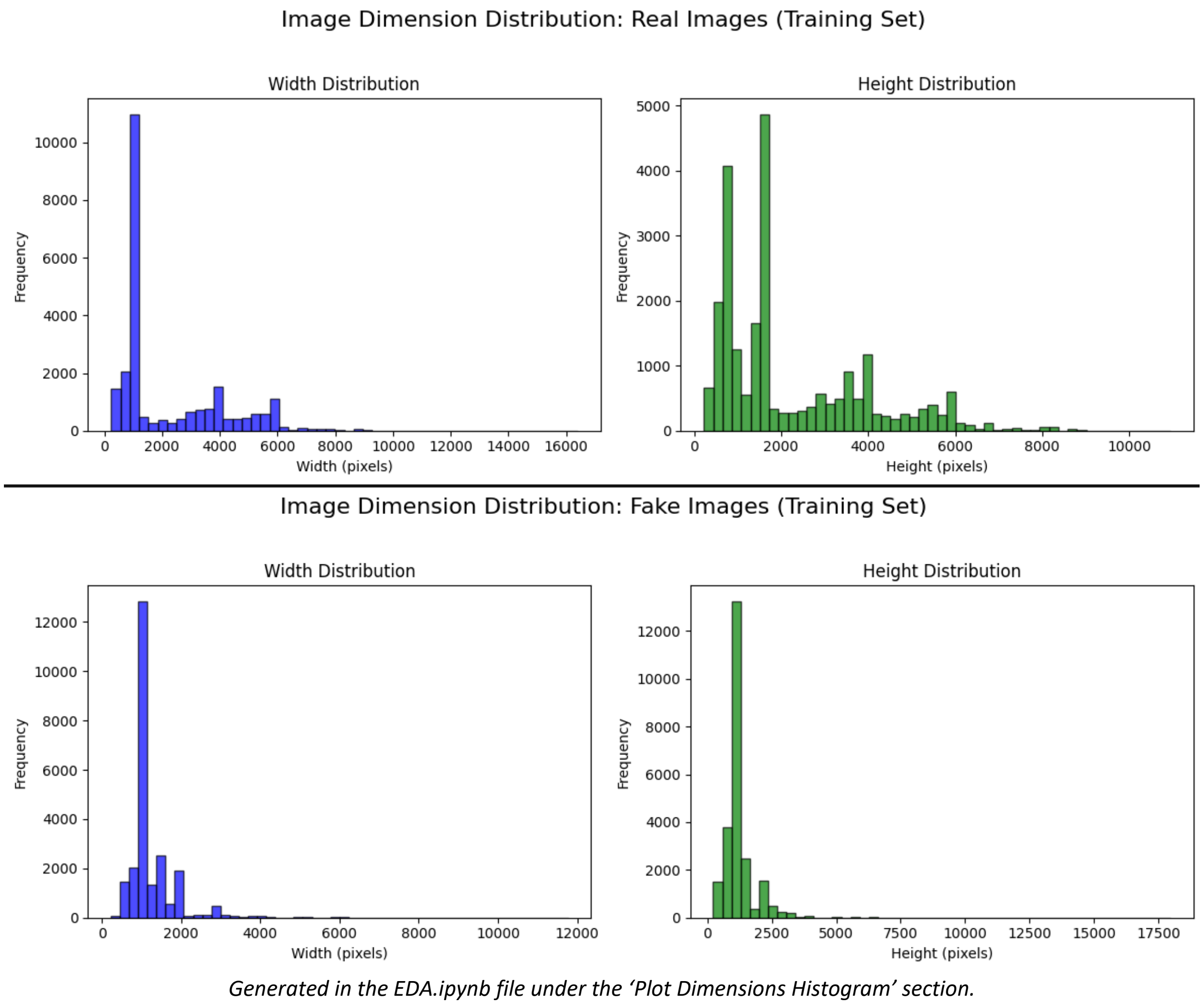
- Fake Images: Average dimensions were approximately 1244x1190 pixels, ranging from 224x224 to 11776x17973 pixels.



```
Analyzing dimensions for real images...
Collected dimensions for 24000 real images.

Analyzing dimensions for fake images...
Collected dimensions for 24000 fake images.

Real Image Stats (Sample):
  Avg Width: 2218.25, Avg Height: 2202.42
  Min Width: 236, Min Height: 227
  Max Width: 16384, Max Height: 10944

Fake Image Stats (Sample):
  Avg Width: 1243.60, Avg Height: 1189.79
  Min Width: 224, Min Height: 224
  Max Width: 11776, Max Height: 17973
```

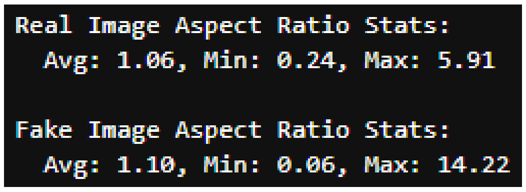
Generated in the EDA.ipynb file under the 'Get Image Dimensions' section.

Distribution: Histograms illustrating the distribution of widths and heights for both classes were generated. These distributions clearly show the widespread and the lack of a standard resolution.



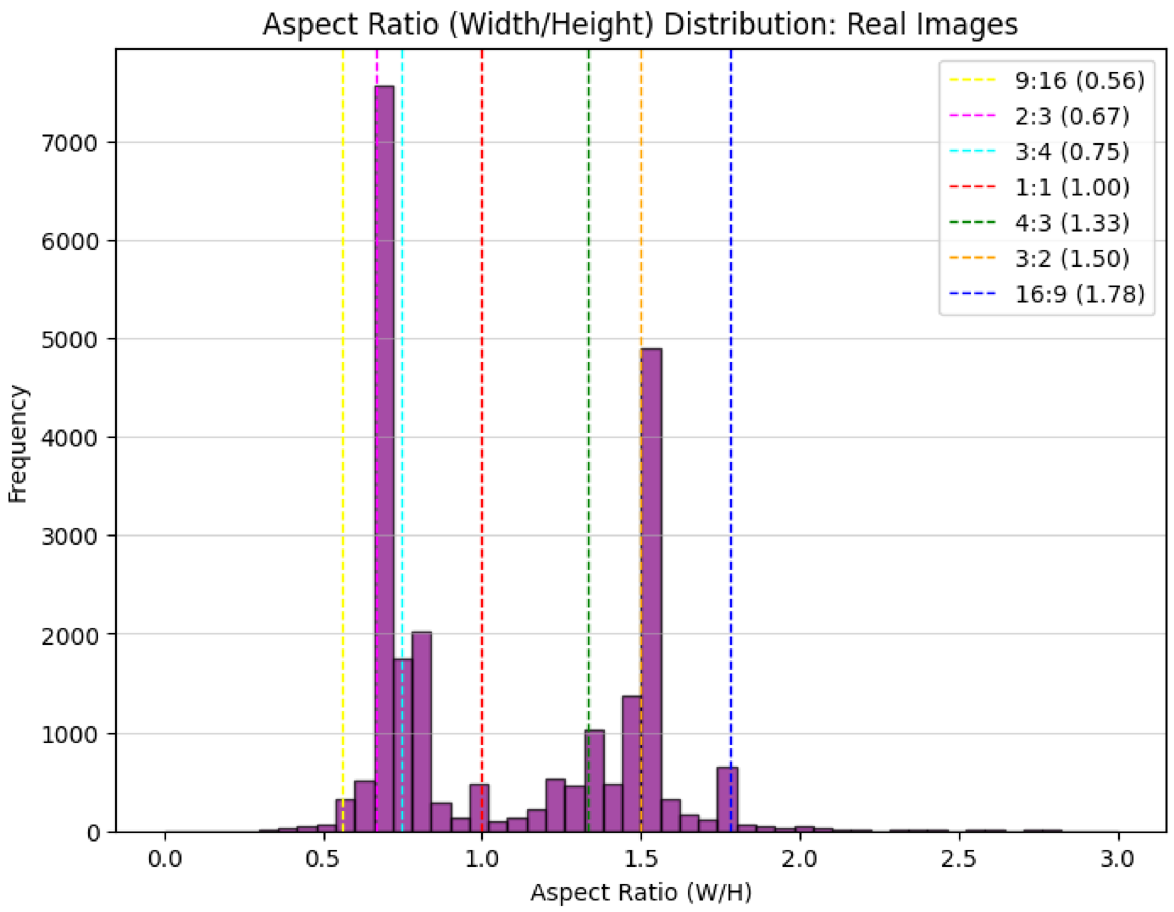
Generated in the EDA.ipynb file under the 'Plot Dimensions Histogram' section.

Aspect Ratio: Analysis of image aspect ratios (width/height) showed similar average values for real (1.06) and fake (1.10) images, though the range was wider for fake images (Min: 0.06, Max: 14.22) compared to real ones (Min: 0.24, Max: 5.91), indicating more extreme outliers. Visualization via histograms revealed a key difference not apparent from the averages alone. Real images displayed a varied distribution with peaks around common portrait (e.g., near 0.67-0.75) and landscape (e.g., near 1.5) ratios. Fake images, however, showed an overwhelmingly dominant peak at the 1.0 (square) aspect ratio, with other ratios being much less frequent.

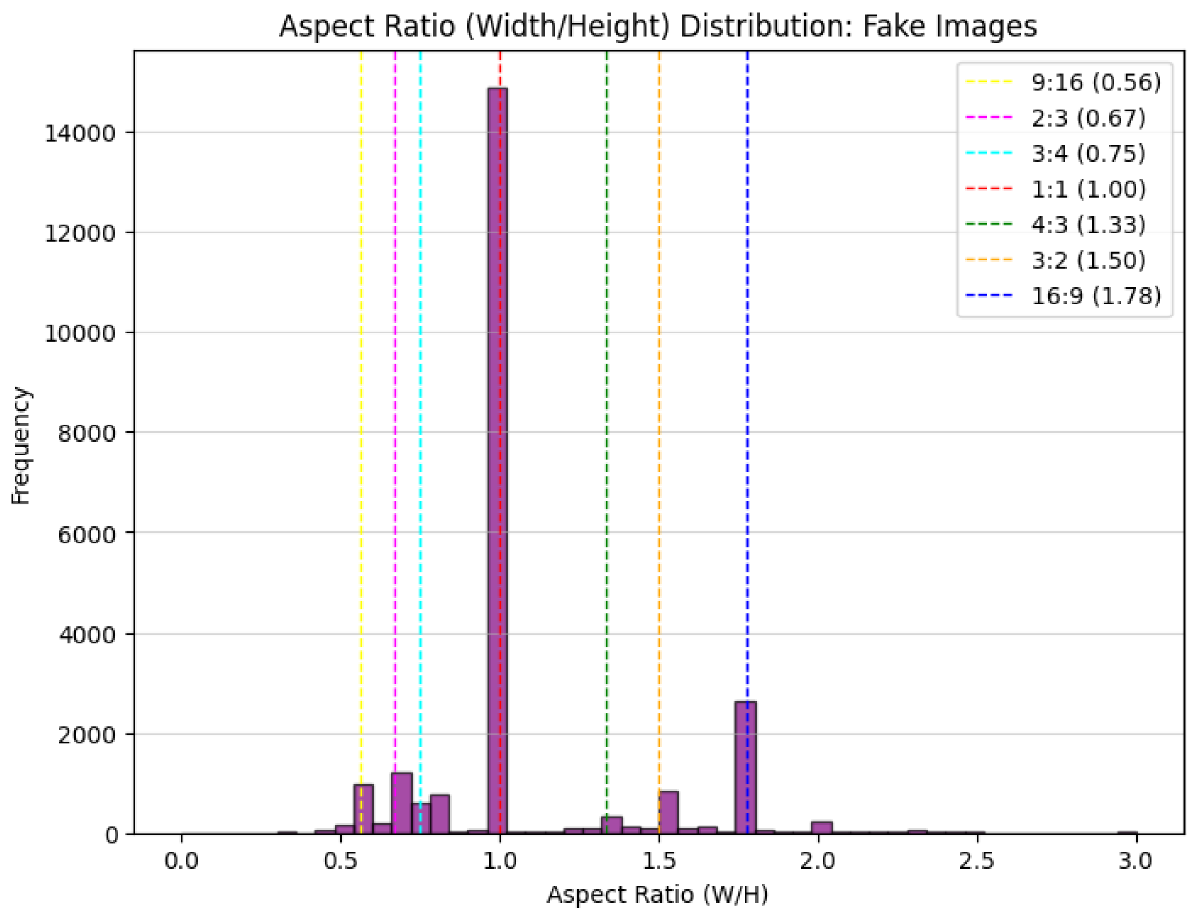


```
Real Image Aspect Ratio Stats:
  Avg: 1.06, Min: 0.24, Max: 5.91

Fake Image Aspect Ratio Stats:
  Avg: 1.10, Min: 0.06, Max: 14.22
```

Generated in the EDA.ipynb file under the 'Calculate Aspect Ratios' section.



Generated in the EDA.ipynb file under the 'Plot Aspect Ratios' section.



Generated in the EDA.ipynb file under the 'Show Sample Images' section.

Preprocessing Implication: These findings guide the necessary preprocessing steps for the dataset. The significant variation observed in image dimensions necessitates resizing all images to a standard size; 224x224 pixels were selected for this purpose. Furthermore, the aspect ratio analysis revealed a notable difference: fake images mostly have a square (1:1) ratio, whereas real images show more varied shapes. This characteristic might provide a distinguishing feature for classification models. The 12 corrupted files identified during the integrity check will be excluded from the dataset before training to prevent errors. While some legitimate large-resolution images required increasing the Pillow library's pixel limit for analysis, the downscaling to 224x224 for model input mitigates potential memory concerns related to these original large files during training. In summary, the EDA indicates that after removing corrupted files and implementing image resizing, the dataset is suitable for the planned modeling, with aspect ratio identified as a potentially significant distinguishing feature.

2. Problem Statement

This project’s goal is addressing the increasingly relevant challenge of distinguishing between real, authentic images and those synthetically AI generated, by training a model that accurately learns the mapping from the input to their correct authenticity class. The core task is formulated as a machine learning problem, given an input image, predict whether it belongs to the 'real' class or the 'fake' class. Success will be evaluated based on standard classification metrics obtained from the test set, including Accuracy, Precision, Recall, and F1-score.

This constitutes a **supervised binary classification** task with the following components:

- Input Data: The input for the model is an image, represented numerically as pixel data (typically height × width × color channels) after appropriate preprocessing.

- Output: The model's output is a prediction corresponding to one of two possible class labels: 'real' or 'fake'.

- Target Variable: As required by the assignment, the target (or dependent) variable for this classification task is the authenticity class of the image, with the discrete values 'real' and 'fake'.

3. Preprocessing

Based on the findings from the Exploratory Data Analysis (EDA), several preprocessing steps were necessary to prepare the image data for input into the machine learning models. These steps focused on standardizing the images and handling data inconsistencies.

Handling Corrupted Data: The data integrity check during EDA identified 12 corrupted image files that could not be loaded. To prevent errors during training and ensure model stability, these 12 specifically identified files were excluded from the dataset lists used for loading data into the models.

Image Resizing: EDA revealed significant variation in image dimensions across the dataset. To create uniform input for the models, all images in the training and testing sets were resized offline to a standard size of 224x224 pixels. This resizing was performed using a separate script (preprocessing.py) employing the LANCZOS resampling filter for high quality downscaling, and the results were saved to new directories. The subsequent model training and testing scripts load data from these directories containing the pre-resized images.

Data Transformation Pipeline: A standard PyTorch transforms.Compose pipeline was applied on-the-fly during data loading in the training and testing scripts. This pipeline included:

- transforms.ToTensor(): This converted the loaded PIL Images (which are in the range [0, 255]) into PyTorch tensors and scaled the pixel values to the range [0.0, 1.0].

- transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]): Following the tensor conversion, the images were normalized using the standard mean and standard deviation values from the ImageNet dataset. This normalization is crucial when using models pre-trained on ImageNet, as they expect input data to be normalized in this specific way.

- transforms.RandomHorizontalFlip(): Random horizontal flips were also utilized in the pipeline to augment the data as it showed to increase model performance.

Data Augmentation: To improve model robustness and generalization, transforms.RandomHorizontalFlip() was applied to the training data during the on-the-fly transformation pipeline. This randomly mirrors the image horizontally with a 50% probability for each image loaded during training. Other augmentations like color jitter were experimented with but did not yield significant improvements and were therefore omitted from the final pipeline.

These preprocessing steps ensured that consistent, clean, and properly formatted data (224x224 normalized tensors) were fed into the classification models.

4. Model Building & Training

Following the preprocessing stage, five different deep learning models were built and trained to address the real vs. fake image classification task. The strategy was to compare performance across different architectural paradigms and scales: standard older CNNs (ResNet family), modern CNNs (ConvNeXtV2 family), and a Vision Transformer (ViT). Specifically, ResNet18, ResNet50, ConvNeXtV2-Tiny, ConvNeXtV2-Base, and ViT-Base/16 were evaluated.

4.1. Model Architectures

ResNet18 & 50: These standard CNN models, based on residual connections, were loaded using torchvision.models. ResNet18 served as a smaller baseline, while ResNet50 represented a deeper, higher-capacity version within the same family.

ConvNeXtV2-Tiny & Base: Representing modern CNN designs inspired by recent research, these models were loaded using the timm library (timm.create_model). ConvNeXtV2-Tiny provided an efficient modern baseline, while ConvNeXtV2-Base offered a much larger capacity within the same architecture family.

ViT-B/16: This Vision Transformer model, loaded via timm, uses a different approach based on self-attention applied to image patches, providing a comparison point between state-of-the-art CNNs and Transformers.

All models utilized transfer learning to leverage knowledge gained from pre-training on the large-scale ImageNet dataset. For each architecture, the default recommended pre-trained weights (Weights.DEFAULT via torchvision or pretrained=True via timm) were loaded. The final classification layer of each pre-trained model was then replaced with a new torch.nn.Linear layer with 2 output units, corresponding to the 'real' and 'fake' classes, ensuring the models were adapted for the specific binary classification task.

4.2. Common Training Setup

A consistent methodology was applied across all models using the PyTorch framework for fair comparison.

Dataset & Loaders: Models were trained using the pre-resized 224x224 pixel images (output from the offline preprocessing script) loaded via datasets.ImageFolder. The transformation pipeline included transforms.RandomHorizontalFlip() (applied only during training), transforms.ToTensor(), and transforms.Normalize() with standard ImageNet means and standard deviations.

Train/Validation Split: The training dataset was split into 90% for training and 10% for validation using torch.utils.data.random_split with torch.manual_seed(SEED) set for reproducible splits across runs. DataLoader instances were used with shuffling enabled for training and disabled for validation. Batch sizes were adjusted based on model size and GPU memory constraints (e.g., 32 for smaller models like ResNet18/ResNet50/ConvNeXtV2-Tiny, reduced to 8 or 16 for larger models like ConvNeXtV2-Base and ViT-B/16).

Loss Function: nn.CrossEntropyLoss was used as the criterion for this binary classification task.

Optimizer: Different optimizers and learning rates were employed based on the model architecture. For the ResNet models (ResNet18 and ResNet50), the optim.Adam optimizer was used with a learning rate of 5e−5. For the ConvNeXtV2 models (ConvNeXtV2-Tiny and ConvNeXtV2-Base) and the ViT-B/16 model, the optim.AdamW optimizer was utilized. AdamW is often recommended for modern architectures and Transformers as it handles weight decay differently from Adam. The learning rate for these models (ConvNeXtV2-Tiny, ConvNeXtV2-Base, and ViT-B/16) was set to 2e−5. A consistent weight decay of 1e−5 was applied across all models and their respective optimizers.

Training Loop & Early Stopping: Models were trained within a loop evaluating performance on the validation set after each training epoch. The validation loss was monitored, and an early stopping mechanism with a patience of 3 epochs was implemented. Training was stopped if the validation loss did not improve for 3 consecutive epochs, preventing significant overfitting and reducing unnecessary training time. The maximum number of epochs was set to 50, but early stopping determined the final training duration for each model. The model state dictionary corresponding to the epoch with the lowest validation loss was saved for later evaluation on the test set.

Hardware: All training was performed using GPU acceleration via CUDA (torch.device('cuda')).

This consistent setup allowed for the fine-tuning of each pre-trained model and the selection of the best performing checkpoint based on validation data, preparing them for the final evaluation.

4.3. Ensuring Reproducibility

Several measures were implemented throughout the model training and evaluation pipeline to ensure results were as reproducible as possible.

Seed Initialization: A global random seed (SEED = 42) was defined and used to initialize the random number generators for PyTorch (torch.manual_seed(SEED)), NumPy (np.random.seed(SEED)), and Python's built-in random module (random.seed(SEED)). This helps in obtaining consistent results across runs for operations that involve randomness, such as model weight initialization (if not using pre-trained weights for all layers initially, though pre-trained weights themselves are fixed), and importantly, the torch.utils.data.random_split for creating training and validation datasets. The seed was also applied to CUDA operations when a GPU or multiple were available (torch.cuda.manual_seed(SEED), torch.cuda.manual_seed_all(SEED)).

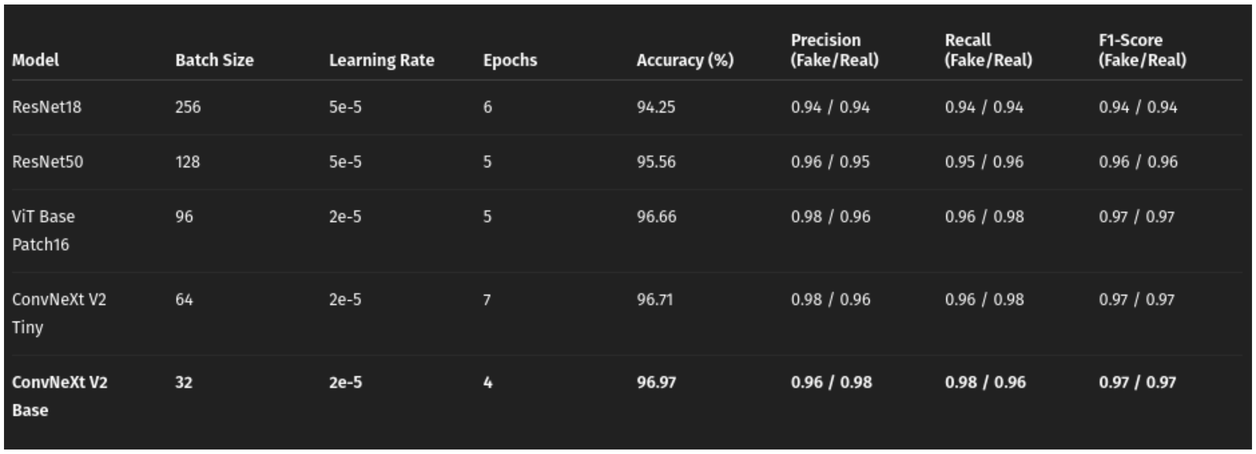
Deterministic CUDA Operations: For GPU-accelerated training, PyTorch's cudnn backend was configured to use deterministic algorithms (torch.backends.cudnn.deterministic = True) and disable benchmarking (torch.backends.cudnn.benchmark = False). While using deterministic algorithms can sometimes slightly impact performance, it is crucial for ensuring run-to-run consistency in results when using CUDA.

Consistent Data Handling: The image transformations applied to the validation and test datasets (transform_test) were strictly deterministic, excluding any random augmentations like RandomHorizontalFlip. This ensures that the model evaluation is performed on the exact same data representation for every run and every model. The pipeline included only transforms.ToTensor() and transforms.Normalize() with standard ImageNet means and standard deviations. Furthermore, when evaluating models on the validation and test sets, the DataLoader was configured with shuffle=False. This guarantees that the data is passed to the model in the same order during each evaluation phase, leading to consistent metric calculations.

Standardized Evaluation Script: The same testing functions (testing_testdataset and testing_traindataset) were used for all models, loading the saved model weights corresponding to the best validation performance and applying the consistent transform_test.
These steps were taken to minimize stochasticity in the training and evaluation process, allowing for more reliable comparisons of the capabilities of the different model architectures tested in this project.

5. Model Evaluation

The performance of the five trained deep learning models were rigorously evaluated on the unseen test set, comprising 20% of the dataset (11,997 images, with 6,000 'fake' and 5,997 'real' images after excluding corrupted files). Standard classification metrics, including accuracy, precision, recall, F1-score, and confusion matrices, were computed to assess and compare their effectiveness in distinguishing between real and AI-generated images. All models were trained until an early stopping criterion (patience of 3 epochs without improvement in validation loss) was met, ensuring that the best performing iteration of each model was used for testing.



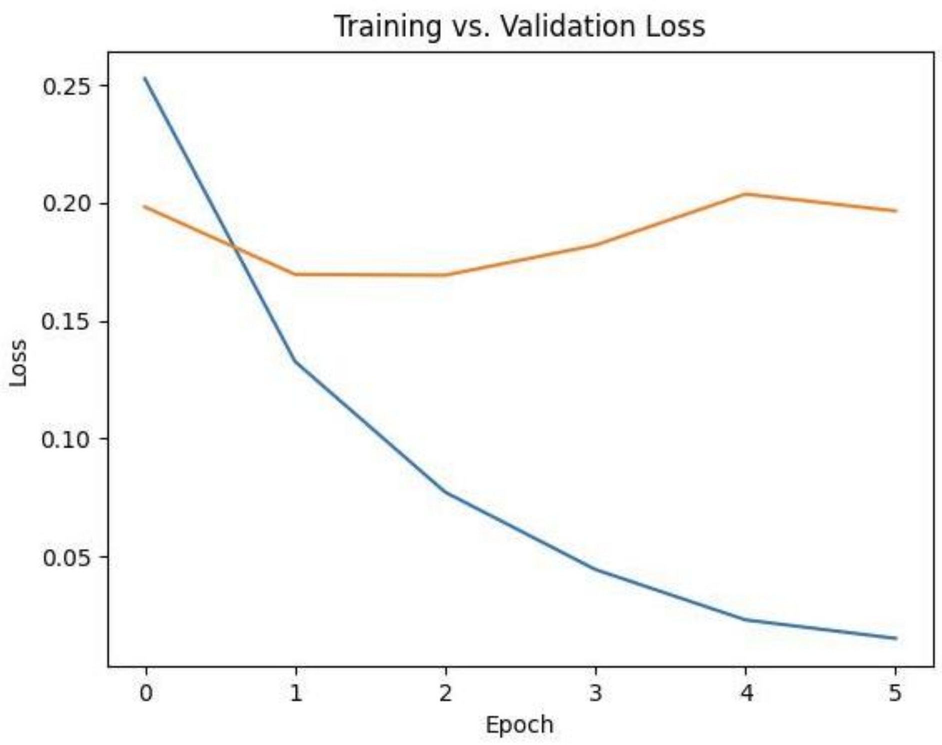
Model	Batch Size	Learning Rate	Epochs	Accuracy (%)	Precision (Fake/Real)	Recall (Fake/Real)	F1-Score (Fake/Real)
ResNet18	256	5e-5	6	94.25	0.94 / 0.94	0.94 / 0.94	0.94 / 0.94
ResNet50	128	5e-5	5	95.56	0.96 / 0.95	0.95 / 0.96	0.96 / 0.96
ViT Base Patch16	96	2e-5	5	96.66	0.98 / 0.96	0.96 / 0.98	0.97 / 0.97
ConvNeXt V2 Tiny	64	2e-5	7	96.71	0.98 / 0.96	0.96 / 0.98	0.97 / 0.97
ConvNeXt V2 Base	**32**	**2e-5**	**4**	**96.97**	**0.96 / 0.98**	**0.98 / 0.96**	**0.97 / 0.97**

Table summarizes key performance metrics and hyperparameters (Generated using AI tools).

The results indicate a generally high level of performance across all models, with accuracies exceeding 94%. The modern CNN architectures (ConvNeXtV2-Tiny and Base) and the Vision Transformer (ViT Base Patch16) outperformed the older ResNet architectures. ConvNeXtV2-Base achieved the highest test accuracy at ~97%.

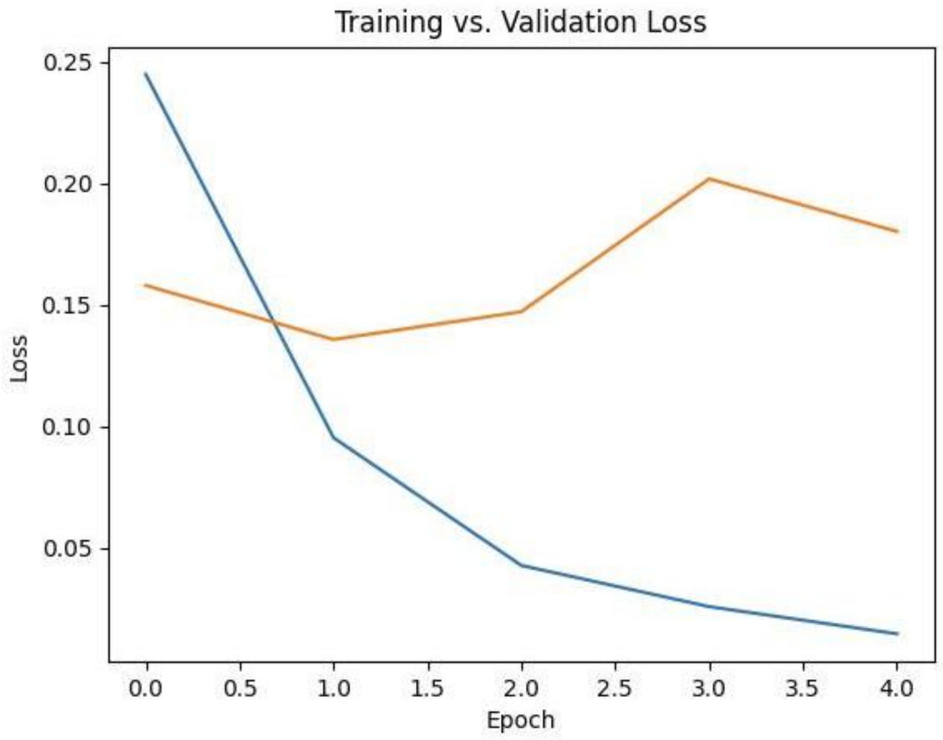
5.1. Detailed Model Performance

ResNet18: This model served as a baseline being the simplest model used in this project; it achieved a test accuracy of 94.25%. The precision, recall, and F1-score were balanced at 0.94 for both ‘resized_fake’ and ‘resized_real’ classes. The confusion matrix showed 339 false negatives (fake images classified as real) and 351 false positives (real images classified as fake) out of 11.997 test images. Training converged relatively quickly, with early stopping triggered after 6 epochs due to no improvement in validation loss.



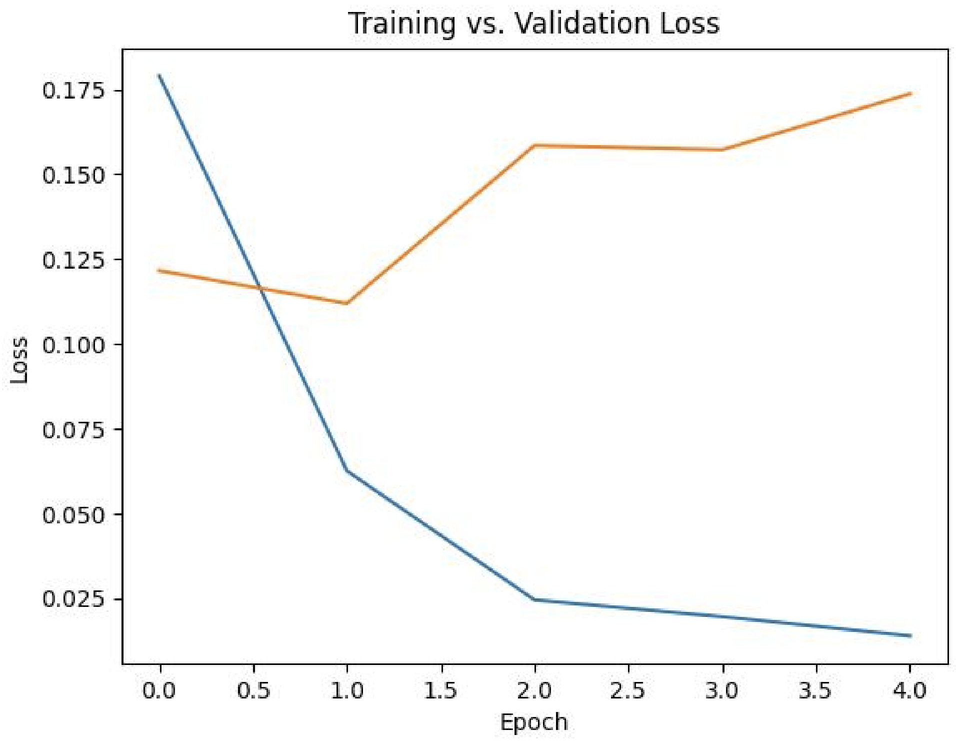
Generated in the model_training_testing.py file of the respective model by using the loss_graph() function.

ResNet50: A deeper model in the ResNet family. It showed an improvement over ResNet18, with a test accuracy of 95.56%. It achieved precision, recall, and F1-scores of 0.96 for both classes on average. The confusion matric showed 310 false negatives and 223 false positives, showing a better ability to distinguish both fake and real images compared to ResNet18. Training was stopped after 5 epochs due to early stopping.



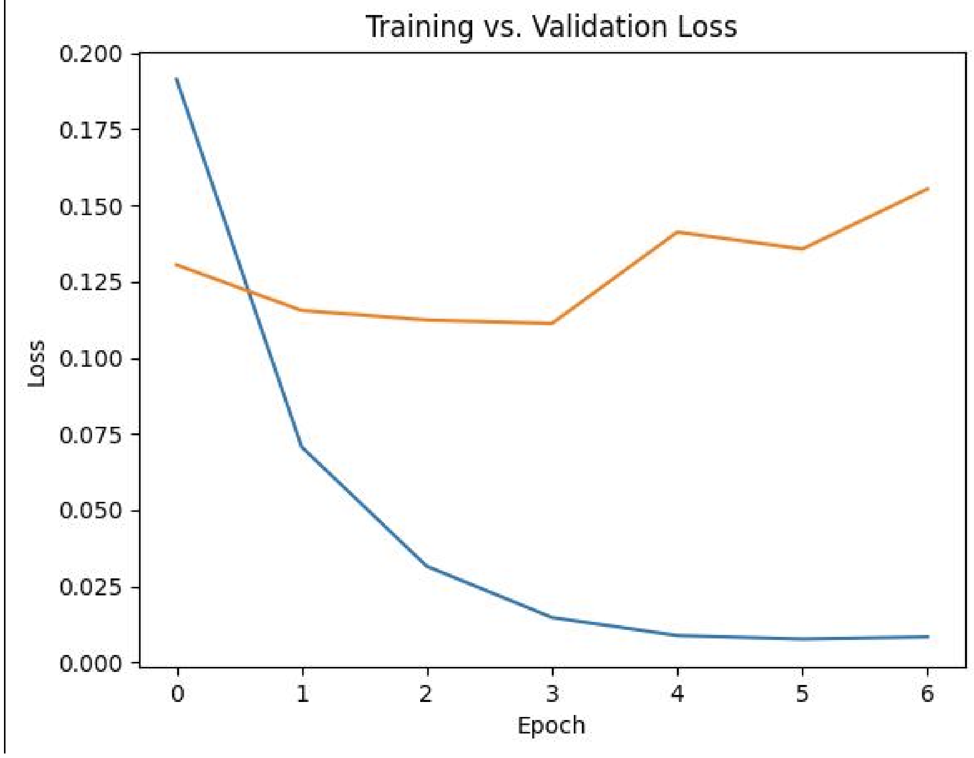
Generated in the model_training_testing.py file of the respective model by using the loss_graph() function.

ViT-B/16: This model performed comparably to the ConvNeXt models, achieving a test accuracy of 96.66% and a macro average F1-score of 0.97. It showed high precision (0.98) for 'resized_fake' images and high recall (0.98) for 'resized_real' images. The confusion matrix presented 259 false negatives and 142 false positives. The model trained for 5 epochs before early stopping. This demonstrates that transformer architectures can also perform well in this image classification task.



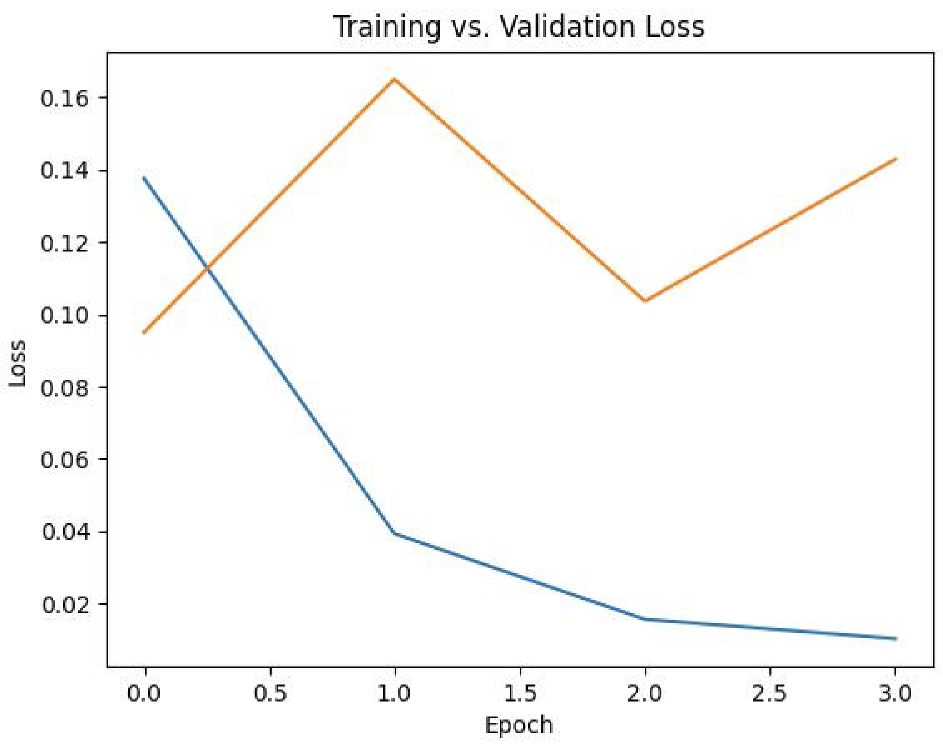
Generated in the model_training_testing.py file of the respective model by using the loss_graph() function.

ConvNeXtV2-Tiny: This model represents a more modern CNN architecture; it achieved a significantly higher test accuracy of 96.71%. The macro average F1-score was 0.97. Precision for ‘resized_fake’ was 0.98 and for ‘resized_real’ was 0.96, while recall was 0.96 and 0.98 respectively. This indicates a slight tendency to be more confident when predicting ‘fake’ and slightly better at identifying all ‘real’ images. The confusion matrix showed 269 false negatives and only 126 false positives. This model trained for 7 epochs before early stopping.



Generated in the model_training_testing.py file of the respective model by using the loss_graph() function.

ConvNeXtV2-Base: A larger model of the ConvNeXt family, yielded the highest test accuracy among all tested models: 96.97%. Its macro average F1-score was also 0.97. It exhibited strong precision for 'resized_real' (0.98) and recall for 'resized_fake' (0.98). The confusion matrix was the most favorable, with only 121 false negatives and 243 false positives. Despite its larger size, training was efficient, with early stopping after just 4 epochs.



Generated in the model_training_testing.py file of the respective model by using the loss_graph() function.

5.2. Comparison of The Models

The evaluation clearly indicates that more recent architectures (ConvNeXtV2 and ViT) outperform the older ResNet models for this specific task of differentiating AI-generated images from real ones. The ConvNeXtV2-Base model demonstrated a slight edge in overall accuracy, though ConvNeXtV2-Tiny and ViT-B/16 were very close behind, all achieving around 96-97% accuracy.

The ResNet models, while robust, plateaued at a lower performance level (94-95.5% accuracy). This suggests that the architectural innovations in ConvNeXt (such as modernized stem and block designs inspired by ViTs) and the attention mechanisms in ViT are beneficial for capturing the subtle artifacts or patterns that distinguish AI-generated images.

Interestingly, the ConvNeXtV2-Base model, despite being larger, converged in fewer epochs (4 epochs), suggesting efficient learning. However, all models benefited from early stopping, preventing overfitting and saving computational resources.

The confusion matrices across all models generally show a balanced number of false positives and false negatives, though the more advanced models significantly reduced the total number of misclassifications. For instance, ConvNeXtV2-Base had the lowest number of fake images misclassified as real (121 instances), which is a lot more important in this task as in a real-world scenario it's much worse to classify fakes images as real than real images as fake.

In conclusion, while all models performed well, the ConvNeXtV2-Base model stands out as the top performer based on test accuracy and the low number of false negatives shown by the confusion matrix. However, the choice of model in a real-world scenario might also consider factors like inference speed and computational cost, where lighter models like ConvNeXtV2-Tiny might be considered due to offering a better trade-off if resources are constrained, especially if the amount of false negatives can be reduced.

6. Feature Importance Analysis

Understanding how deep learning models make predictions is crucial for validating their decisions and identifying potential biases. This section explores the internal mechanisms of the trained models, analyzes feature importance using Grad-CAM for the CNN-based models, and discusses why Grad-CAM is not applicable to the Vision Transformer (ViT-B/16), proposing an alternative method for transformers.

6.1. Overview of Model Mechanisms

When a CNN processes an image, it uses kernels to perform convolution operations, sliding over the image and computing dot products to extract features like edges, textures, or colors. These kernels produce feature maps that highlight the presence of specific patterns. For distinguishing real vs. AI-generated images, these features might include natural textures in real images or artifacts like irregular lighting or symmetry in fake images.

Activation functions introduce non-linearity, allowing the model to learn complex patterns. For instance, ResNet18 and ResNet50 use ReLU (Rectified Linear Unit), which zeros out negative values to focus on significant features, while ConvNeXtV2 models use GELU (Gaussian Error Linear Unit), which is smoother and often better for modern architectures. In our binary classification task, the output layer uses CrossEntropyLoss, which internally applies Softmax to produce probabilities for the 'real' and 'fake' classes.

Pooling layers reduce the spatial dimensions of feature maps, summarizing features within a region (e.g., max pooling takes the maximum value in a window, while average pooling computes the mean). This makes the model robust to small shifts in the image, which is useful for handling variations in both real and fake images. Finally, fully connected layers interpret the filtered features to make the final classification decision, combining all extracted information to determine the most likely class.

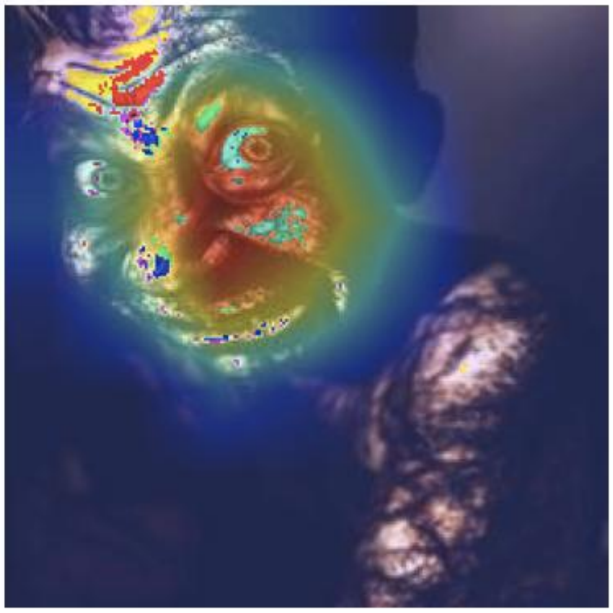
6.2. Grad-CAM Visualizations for CNN Models

To better understand what the CNN models focus on when making predictions, we can use Grad-CAM (Gradient-weighted Class Activation Mapping). Grad-CAM works by computing the gradients of the predicted class score with respect to the feature map's final convolutional layer, using these gradients to weigh the importance of each feature map. The result is a heatmap highlighting the most relevant regions in the image, with warmers colors (red, yellow) indicating higher importance and cooler colors (blue, green) indicating lower importance.

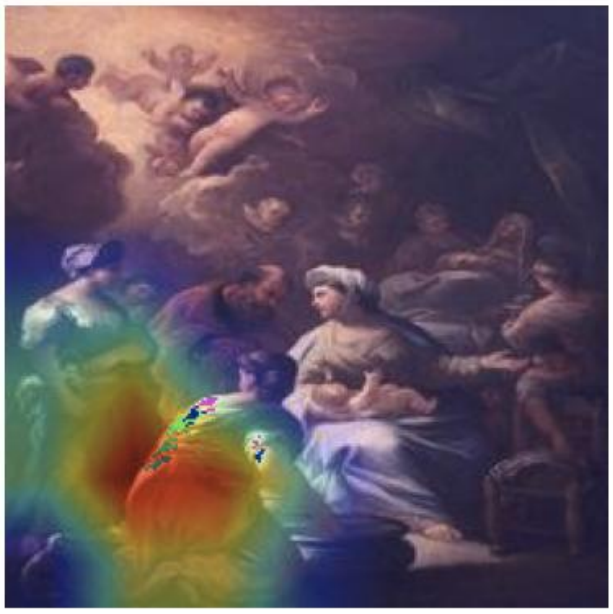
We generated Grad-CAM heatmaps for four CNN models (ResNet18, ResNet50, ConvNeXtV2-Tiny and ConvNeXtV2-Base) using two images: a ‘fake’ AI-generated face (0101.jpg) and a ‘real’ classical painting with multiple figures (1010.jpg). These images were kept consistent across models for a fair comparison.

ResNet18: For the fake image, the heatmap shows strong attention on the eyes and mouth, suggesting it detects artifacts like unnatural textures or shapes in these areas, common in AI-generated faces. In the real image, the attention is focused on two women, one of which has her back towards the viewer, indicating that perhaps ResNet18 struggles to focus on more meaningful features such as the faces or background.

Grad-CAM Visualization of attention

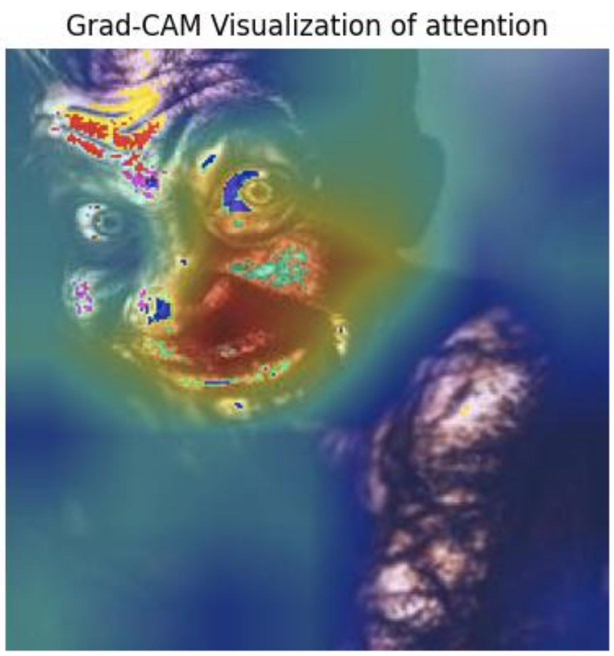


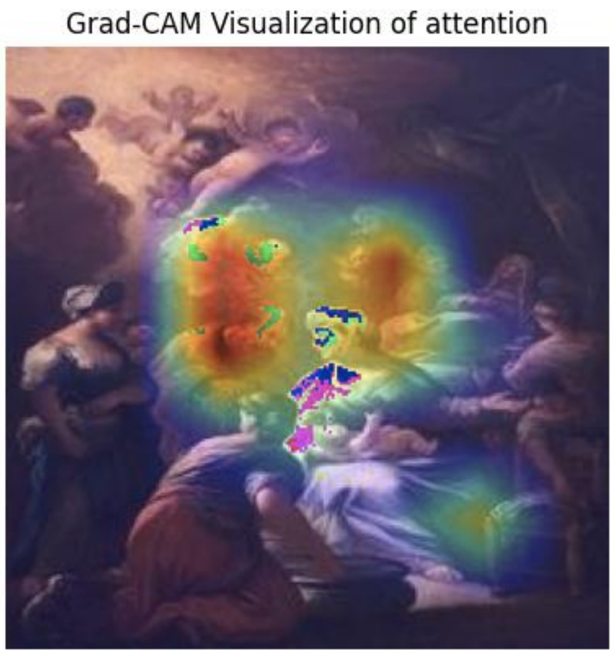
Grad-CAM Visualization of attention



Generated in the grad-cam.py file of the respective model.

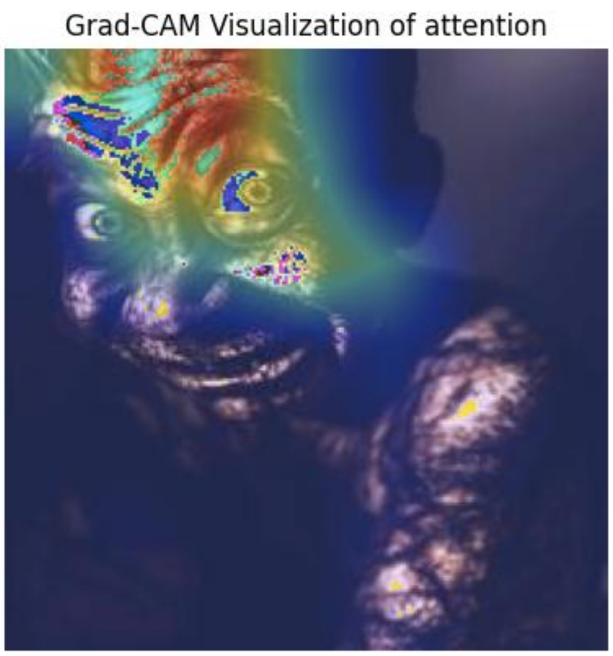
ResNet50: This deeper model shows more refined attention. In the fake image, whilst still focusing on the face, it is more selective as to which features of the face it focuses on, suggesting better detection of specific artifacts. For the real image, attention focuses on the women's faces and upper bodies in the center of the painting, showing also a better focus on more relevant artifacts.

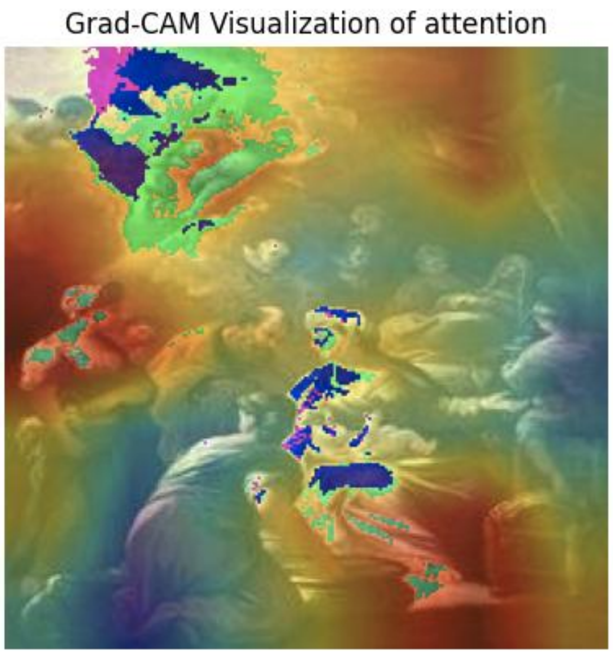




Generated in the grad-cam.py file of the respective model.

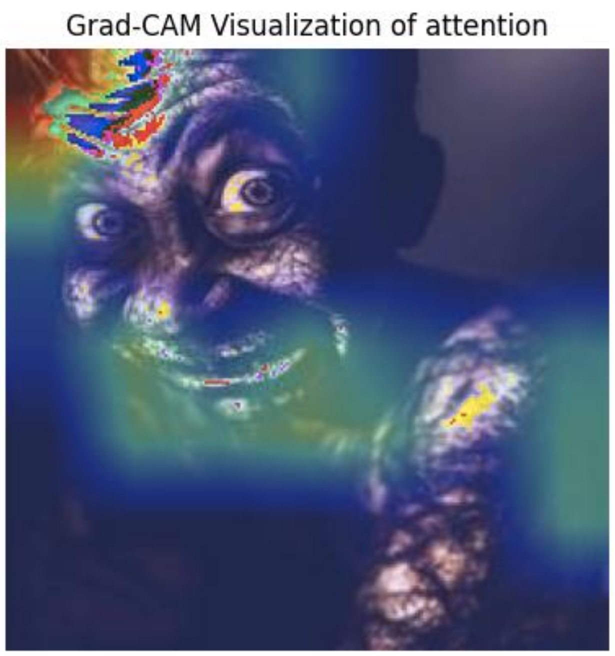
ConvNeXtV2-Tiny: This model exhibits an even more localized focus in the fake image, preferring to pay more attention to the upper part of the side and also selecting certain features in the same. In the real image, attention is spread out throughout the entire painting, suggesting the model considers the overall composition of the image.

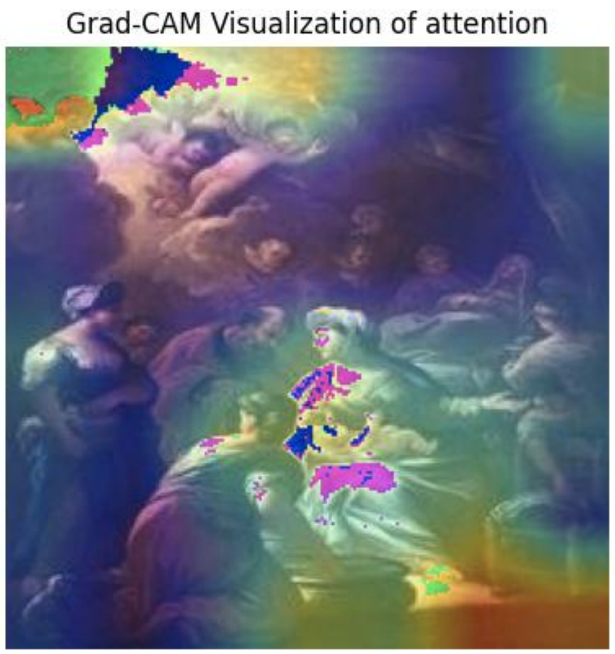




Generated in the grad-cam.py file of the respective model.

ConvNeXtV2-Base: The top performer shows interesting behaviour, the areas it focuses on are smaller than previous models, possibly due to it honing on very specific parts of the images to classify them. In the fake image we clearly see that the main focus were small areas in the forehead and the jaw. In the real image attention is scattered and on the corners of the image, suggesting its catching details in the background and other structural cues present.





Generated in the grad-cam.py file of the respective model.

6.3. Why Grad-Cam Doesn't Work for ViT-B/16

Grad-CAM is a common tool to visualize convolutional neural networks by showing highlights of which parts of the picture/image that have the highest effect on the class prediction. To create the heatmap it uses feature maps from the last convolution layer weighted by gradients. This works because convolutional neural networks maintain its spatial structure and then highlight the key picture/image regions

In comparison, Vision Transformers process images as sequences of patches and utilize self-attention mechanisms to model the relationships between these patches. Instead of relying on gradients, Visual Transformers often use attention maps or attention rollout techniques to analyze the attention weights between the [CLS] token and patches. With the information gathered by these techniques we are able to visualize where the model focuses its attention. These methods emphasize the flow of information through the attention layers rather than emphasizing the gradient-based information. Then it will need to reconstruct path influence since Vision Transformers don't preserve the spatial structure like convolutional neural networks do. Since Vision Transformers do not inherently preserve spatial structure like convolutional neural networks, each patch will need to be reconstructed to determine which parts of the input contribute most to the model's prediction.

To understand feature importance in ViT-B/16, an alternative method like Attention Rollout can be used. Attention Rollout aggregates the attention weights across all transformer layers to estimate the contribution of each input patch to the final prediction. This method traces the flow of attention from the input to the output, producing a heatmap-like visualization that highlights important patches. For our task, Attention Rollout could reveal whether ViT-B/16 focuses on specific patches, or global context.

6.4. Insights

The Grad-CAM heatmaps reveal that all CNN models focus on facial features in fake images, likely detecting artifacts like unnatural textures or shapes, which aligns with the EDA finding that fake images often have distorted features (e.g. text, hands, etc). The heatmaps confirm that the models are indeed "looking" at the image content to make decisions. The differences in attention patterns across models also hint at why some models might be more successful than others.

It is also important to acknowledge that this Grad-CAM analysis is based on a visual inspection of heatmaps for only two sample images per model. While indicative, these observations may not generalize to all images or all types of AI-generated content. A more comprehensive analysis would involve evaluating Grad-CAM results over a larger and more diverse set of images from the test set and potentially quantifying the attention patterns. Furthermore, Grad-CAM provides a high-level view of feature importance from the final convolutional layer and does not fully show the complex interplay of features throughout the network.

7. Real-World Applicability

The ability to reliably distinguish between authentic and AI-generated imagery, as demonstrated by the models developed in this project, has significant real-world implications. The ConvNeXtV2-Base model, achieving approximately 97% accuracy and notably low false negative rates (misclassifying fake images as real), shows particular promise for practical deployment in various critical domains, if resources and computational power are freely available.

7.1. Applications in Real-World Scenarios

Deepfake detection, Misinformation Prevention

The rise of deepfakes, AI-generated images or videos often depicting manipulated scenarios pose a significant threat to public trust. This model offers ways to help prevent the spread of misinformation about politicians or people of influence by verifying if the picture has been altered or AI generated to taint the subject's image. It can also help stop propaganda about countries and fake news by ensuring that visuals accompanying stories are genuine before it ever reaches a reporter's desk. Within academia and scientific community, the model could be used to verify the authenticity of images in papers or articles.

Forensic Analysis and Law Enforcement

Forensic teams could use the model to verify whether an image is genuine or manipulated. This could help distinguish between real evidence and fabricated or altered visuals.
During trials, the model could authenticate digital image evidence, providing courts with a tool to assess its validity. This application could be pivotal in cases where visual proof is contested.

Intellectual Property and Commercial Use

Here the model could be used to ensure that the images used commercially are real or generated by an AI. This would make people able to determine if the commercials are using real photos or are trying to deceive the general public by using generated images. On platforms like Instagram or TikTok, the model could detect AI-generated or altered images that promote unrealistic beauty standards or misinformation. By flagging such content, it could help protect younger audiences from harmful influences.

7.2. Technical Considerations for Deployment

The model's high performance (e.g. 96.97% accuracy with ConvNeXtV2-Base) suggests strong potential for real-world use, but several factors must be considered:

- Generalizability: The model was trained on images from Stable Diffusion, Midjourney, and DALL-E. Its effectiveness might decrease with images from newer or different AI generators not represented in the dataset. Continuous retraining with diverse, updated data is necessary to maintain accuracy.

- Image Variability: Performance could vary with factors like resolution or content type (e.g., faces vs. landscapes), as noted in the EDA's findings on dimension diversity. Testing across varied conditions would ensure robustness.

- Computational Efficiency: Real-time deployment on high-traffic platforms requires efficient inference. While ConvNeXtV2-Base excels in accuracy, lighter models like ConvNeXtV2-Tiny might be preferred in resource-constrained settings, balancing performance and speed.

7.3. Ethical and Societal Implications

Deploying this model raises important ethical questions:

- Bias and Fairness: The dataset's balance (e.g. 30,000 real vs. 30,000 fake images) is a strength, but biases could emerge if certain AI generators or image types are underrepresented. This could lead to uneven performance across contexts.

- Creative Impact: Overuse might flag legitimate AI-generated art as "fake," stifling creativity. Guidelines should differentiate between harmful fakes and benign artistic uses.

7.4. Summary and Future Advancements

Ethical considerations remain paramount. While beneficial for combating fraud and misinformation, the deployment of AI image detection must be balanced against concerns of stifling legitimate creative uses of AI and other possible misuses. Future advancements should therefore focus not only on improving detection accuracy and efficiency but also on developing transparent, explainable, and ethically responsible AI systems. This includes robust frameworks for model governance, bias detection, and ensuring that these tools empower rather than impede digital authenticity and creativity. Ultimately, the goal is to ensure these detection models serve as reliable and fair tools of visual truth in an increasingly synthetic digital world.

8. References

Ardi, M. (2025, May 6). The CNN That Challenges ViT. *Towards Data Science*. https://towardsdatascience.com/the-cnn-that-challenges-vit/

Best Practices Image Preprocessing in Image Classification | Keylabs. (2024, August 14). Keylabs: Latest News and Updates. https://keylabs.ai/blog/best-practices-for-image-preprocessing-in-image-classification/

Bhandari, M. (2024, March 9). Grad-CAM: A beginner's Guide. *Medium*. https://medium.com/@bmuskan007/grad-cam-a-beginners-guide-adf68e80f4bb

ConvNeXt V2. (n.d.). Retrieved May 19, 2025, from https://huggingface.co/docs/transformers/model_doc/convnextv2

Furnieles, G. (2022, September 8). Sigmoid and SoftMax Functions in 5 minutes. *TDS Archive*. https://medium.com/data-science/sigmoid-and-softmax-functions-in-5-minutes-f516c80ea1f9

GJLi. (2023, March 19). A Short Note On Visualizing Attention of Vision Transformer (ViT). *Medium*. https://medium.com/@UlamacaLEE/a-short-note-on-visualizing-attention-of-vision-transformer-vit-a6b67377a914

How is random flipping used in data augmentation? (n.d.). Retrieved May 19, 2025, from https://milvus.io/ai-quick-reference/how-is-random-flipping-used-in-data-augmentation

How to handle overfitting in PyTorch models using Early Stopping. (01:07:03+00:00). GeeksforGeeks. https://www.geeksforgeeks.org/how-to-handle-overfitting-in-pytorch-models-using-early-stopping/

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., & Xie, S. (2022). *A ConvNet for the 2020s* (No. arXiv:2201.03545). arXiv. https://doi.org/10.48550/arXiv.2201.03545

Max Pooling. (2019, May 17). DeepAI. https://deepai.org/machine-learning-glossary-and-terms/max-pooling

Oleszak, M. (n.d.-a). *Deep Learning for Images with PyTorch—DataCamp Learn*. Retrieved May 19, 2025, from https://app.datacamp.com/learn/courses/deep-learning-for-images-with-pytorch

Oleszak, M. (n.d.-b). *Evaluating object recognition models | PyTorch*. Retrieved May 19, 2025, from https://campus.datacamp.com/courses/deep-learning-for-images-with-pytorch/object-recognition?ex=5

Piantic, H. (n.d.). *Vision Transformer (ViT): Visualize Attention Map*. Retrieved May 20, 2025, from https://kaggle.com/code/piantic/vision-transformer-vit-visualize-attention-map

Pykes, K. (n.d.). *AdamW Optimizer in PyTorch Tutorial*. Retrieved May 19, 2025, from https://www.datacamp.com/tutorial/adamw-optimizer-in-pytorch

Python Image Resize With Pillow and OpenCV. (n.d.). *Cloudinary*. Retrieved May 19, 2025, from https://cloudinary.com/guides/bulk-image-resize/python-image-resize-with-pillow-and-opencv

Renotte, N. (n.d.). *Build a Deep CNN Image Classifier with ANY Images—YouTube*. Retrieved May 19, 2025, from https://www.youtube.com/watch?v=jztwpsIzEGc

ResNet. (n.d.). Retrieved May 19, 2025, from https://huggingface.co/docs/transformers/model_doc/resnet

Tomar, N. (n.d.). *GradCAM Implementation in PyTorch—MobileNetv2 Heatmap Visualization | OpenCV - YouTube*. Retrieved May 19, 2025, from https://www.youtube.com/watch?v=eLQZrNYqjNg

TorchVision Object Detection Finetuning Tutorial—PyTorch Tutorials 2.7.0+cu126 documentation. (n.d.). Retrieved May 20, 2025, from https://docs.pytorch.org/tutorials/intermediate/torchvision_tutorial.html#visualizing-the-predictions

Tuychiev, B. (n.d.). *Adam Optimizer Tutorial: Intuition and Implementation in Python*. Retrieved May 19, 2025, from https://www.datacamp.com/tutorial/adam-optimizer-tutorial

Vision Transformer (ViT). (n.d.). Retrieved May 19, 2025, from https://huggingface.co/docs/transformers/model_doc/vit

Wang, Y., Deng, Y., Zheng, Y., Chattopadhyay, P., & Wang, L. (2025). Vision Transformers for Image Classification: A Comparative Survey. *Technologies*, *13*(1), Article 1. https://doi.org/10.3390/technologies13010032

Woo, S., Debnath, S., Hu, R., Chen, X., Liu, Z., Kweon, I. S., & Xie, S. (2023). *ConvNeXt V2: Co-designing and Scaling ConvNets with Masked Autoencoders* (No. arXiv:2301.00808). arXiv. https://doi.org/10.48550/arXiv.2301.00808