JOURNAL OF
EMERGING INVESTIGATORS

Article

Optimizing AI image detection using a Convolutional Neural Network model with Fast Fourier Transform

Pranav Gupta[1], Larry McMahan[2]

[1] Monta Vista High School, Cupertino, California

[2] Department of Computer Science & Engineering, Aspiring Scholars Directed Research Program, Fremont, California

SUMMARY

Recent progress in generative artificial intelligence (AI) technology has made it increasingly difficult to tell whether an image is AI-generated or real. Current AI detection methods rely solely on training a model through convolutional neural networks (CNNs) or using the U-net architecture with Intersection-over-Union as an evaluation metric. However, these approaches are not accurate enough to obtain sufficient detail from spatial (traditional) and frequency domain features, missing discrete patterns in edges and textures. Our study aimed to determine if a tool incorporating CNN and Fast Fourier Transform (FFT) can improve the detection of AI-generated images. We hypothesized that using a CNN model and frequency domain features that leverage FFT would lead to improvements in AI-generated image detection. Our study presented a machine learning model that utilizes FFT to transform images from the spatial to the frequency domain, where subtle features can be extracted. This process involved training the model on a dataset comprised of 100,000 training images and 20,000 testing images. The model preprocesses images through FFT and merges the images into CNN logic to achieve 93.30% accuracy while handling both colored and grayscale inputs. Our results demonstrated that the integrated approach performed better than the CNN-only approach, showing 10.06% lower loss, 3.27% higher accuracy, 7.79% higher true positive rate, and 3.48% higher F1-score. This research offers an improved method for distinguishing between AI-generated and genuine images, which will be highly impactful as generative AI produces more malicious fake news and video attempts.

INTRODUCTION

The rapid advancement of artificial intelligence (AI) technologies has led to the proliferation of many highly realistic, AI-generated images. Text-to-image generation is a way of using AI with machine learning to create images from text descriptions, which can be done on a large scale (1). In 2023, people using DALLE-2, an AI tool created by OpenAI that can produce images in different styles, generated over 34 million images per day, with users also making use of text-to-image platforms and algorithms to generate over 15 billion images that year (2). For context, that's how many images photographers captured over 150 years, from 1826 to 1975 (2). Text-to-image generation is not necessarily deleterious. AI-generated images can offer benefits, primarily by accelerating the creative process, reducing costs, and enabling new forms of artistic expression (3). They can also be used for rapid prototyping, generating diverse visual content, and even assisting artists in their workflow (3). However, numerous concerns and negative effects also arise from AI-generated images since they can blur the lines between authentic and fabricated content. For example, a real-world connection to this problem is what happened on the morning of May 22, 2023, when a false AI-generated image claimed that there was an explosion inside the United States Pentagon (4). Public media sources then spread that image, causing financial markets to fall (5).

Several methods have been utilized to detect AI-generated images using convolutional neural networks (CNNs) in pre-trained models such as ModelNet and ResNet50 (6). CNN is a type of deep learning algorithm, often implemented for media processing. It utilizes a hierarchy of layers, including convolution, pooling, normalization, and fully connected layers, to transform an input image into an output volume with class scores (7). The term "class scores" refers to the numerical outputs from the final layer of a CNN, where each number represents the model's confidence for which specific class (category) the image might belong to (8). In a typical CNN, the whole network has many layers that become smaller as information is passed further through the network. One recent study using a CNN-based method trained on real and AI-generated images achieved up to 92.98% classification accuracy (9). Another benchmark study using a CNN-based method trained on a similar dataset achieved an overall classification accuracy of 93.67% when detecting real versus AI-generated images (10). The differences in results between these two studies are due to how the CNNs themselves were implemented and differences in specific parts.

Alternative methods discuss approaches involving a dataset with generators and pixel prediction (6, 11). Other works approach this differently by using U-net architecture and Intersection-over-Union (12). This approach is different compared to using generators and pixel prediction because the U-net architecture is a neural network architecture that uses an encoder-decoder structure to segment images and the Intersection-Over-Union metric is an evaluation metric used to measure accuracy by calculating the amount of overlap between two bounding boxes (12). It is mainly used for evaluation and not specifically for binary tasks, where AI-generated image detection versus real image detection is a binary task (12). According to a study, the average accuracy obtained when using the U-net architecture and Intersection-over-Union with frequency domain features is 83% (12). However, one of the current limitations of these approaches

https://doi.org/10.59720/24-270

is that enough detail from traditional and frequency domain features isn't extracted, and thus, AI-generated images pose challenges for detection systems that focus only on spatial domain analysis (13). This is because spatial domain methods analyze visible patterns such as edges and textures, but generative models often leave subtle artifacts in the frequency domain through patterns that are invisible to the naked eye but detectable through frequency domain analysis and not only spatial domain analysis. Without extracting these details, detectors may miss signs of manipulation (14).

The limitations of these AI-generated image detection methods prompt a key question about whether the integration of CNNs and Fast Fourier Transform (FFT) can efficiently detect AI-generated versus real images to combat unwanted image manipulation. FFT is a computational tool used in signal processing to convert time-domain data into frequency-domain data, enabling the analysis of a signal's frequency components (12). It is essential for frequency analysis, data compression, and pattern detection tasks. Pieces of processed information are found from the analysis of frequency domain features. These domain features are involved with wavelengths and mathematical functions that help determine the information (15). Frequency domain analysis reveals patterns and anomalies that are not easily detectable in the spatial domain (12). FFT has been shown to capture texture and structural information that AI generation processes might affect (12). By converting images into the frequency domain, FFT allows the CNN to analyze these additional features, likely improving the model's performance (12). Simply put, FFT takes a discrete signal from an input and uses that information in an algorithmic approach. This results in a model that uses the FFT to access more information points from images.

In this study, we investigated whether an approach using CNN and FFT can improve the detection of AI-generated images. We hypothesized that using a CNN model and frequency domain features that leverage FFT would improve AI-generated image detection. Our hypothesis is based on the ability of the FFT to transform images from the spatial domain to the frequency domain, where subtle features can be used (12). The developmental goal is to let users autonomously, accurately, and uniquely distinguish between AI-generated and genuine images through a dual-pronged approach. Key subgoals to incorporate are robustness (training and testing the model on a variety of dataset sizes), adaptability (accepting multiple image sizes and collapsing excess dimensions), and functionality (producing accurate classification and optimizing training). When comparing the CNN-only approach to the CNN-FFT approach, evaluation metrics such as accuracy and loss showed promise in this implementation and the extent to which frequency domain features can enhance AI-generated image detection. This study highlights the limitations of relying solely on spatial domain analysis for detecting AI-generated images, since such methods often

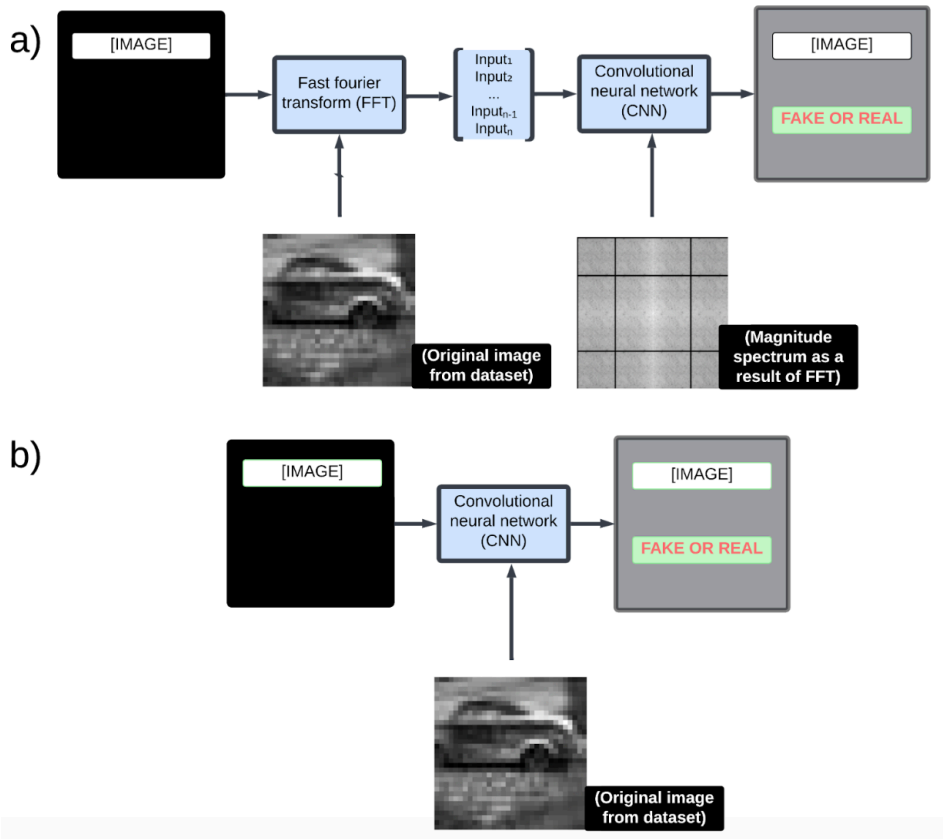

Figure 1:Flowcharts of CNN-only and CNN-FFT models. a) Illustrates the process from the original image to classification using CNN with FFT. The image is passed through FFT before the additional information is combined with the CNN logic for a result to then be outputted. b) Illustrates the process from the original image to classification using CNN without FFT. The image is passed directly to the CNN for a result to then be outputted.

https://doi.org/10.59720/24-270

overlook subtle yet informative frequency-based artifacts. It is plausible that integrating FFT with CNNs provides a more comprehensive representation of image data, enabling more accurate classification and showing the value of dual-domain analysis in enhancing model robustness and reliability.

RESULTS

By integrating FFT with a CNN, we aimed to enhance AI-generated image detection accuracy by using a dual-domain approach integrating FFT and CNNs. We performed this study by using a common dataset on two separate models: a CNN without the use of FFT (CNN-only) and a CNN with the use of FFT (CNN-FFT) **(Figure 1)**. The CNN-only method focuses solely on spatial patterns, while the CNN-FFT method enhances detection by incorporating frequency-based features. In CNN-FFT, an input image undergoes FFT, transforming it into a frequency domain representation. These features are then combined with the original spatial domain inputs and fed into the CNN for classification.

Compared to the CNN-only model, the CNN-FFT model showed an increase in average accuracy by 3.27% and a decrease in average loss by 10.06% **(Table 1)**. The CNN-FFT model also showed an increase in recall, the ratio of all actual positives that were classified correctly as positives, by 7.79%, and an increase in F1-score, the harmonic mean of precision and recall, by 3.48% (16) **(Table 2)**. Data from confusion matrices show better performance from the CNN-FFT model due to a smaller total count of false negatives and false positives **(Figure 2)**. Results indicated that FFT affects the time complexity and the use of computational resources. FFT has a time complexity of O(N×logN), where 'O' is the growth rate of a function and 'N' is the number of input data points (17). This reduced memory requirements by sparsifying features in the frequency domain. This difference applied only to the training phase and roughly reflected the proportional time difference for other stages, such as graphical visual generation or metric calculation. This study's results were useful in the back-end creation of a website, in an early beta testing phase, that outputs whether an image is AI-generated or not with the CNN-FFT method **(Figure 3)**.

DISCUSSION

Our research investigation was primarily aimed at determining if the inclusion of FFT would improve the ability of the model to detect AI-generated images. Comparing the models, we observed that the accuracy and F1-score, as well as recall, were improved when using a CNN-FFT model. These enhancements are due to FFT's ability to transform images into the frequency domain, revealing beneficial image features. The frequency components captured by the FFT enhance the model›s ability to identify specific patterns associated with AI-generated images, resulting in improved classification accuracy and better discrimination between AI-generated and real pictures.

Our machine learning approach using frequency domain features clearly enhanced the detection of images. A large dip in loss from 31.28% to 21.22% and a jump in accuracy from 88.19% to 91.46% shine a positive light on this approach, with concrete improvements in other aforementioned metrics. Although both models have similar logic, the main difference is the addition of the FFT aspect to the CNN-FTT model to ensure the validity of this study and its results by eliminating the occurrence of confounding variables. Overall, the best result this study produced was on the 120,000-image dataset with the CNN-FFT model. The following metrics were achieved: 93.30% accuracy, 17.25% loss, 93.20% precision, 93.43% recall, and 93.31% F1-score. Interestingly, we observed a slight decrease in precision by 1.24%. This was because it is easier to have a false negative than a false positive.

Similar research has been done in this area through other studies (6, 9-14). The first study achieved 92.98% accuracy using a CNN focusing on spatial-domain features and employed Gradient-weighted Class Activation Mapping to show that the model often based its predictions on small visual imperfections in the backgrounds rather than the main subjects (9). This caused it to learn to detect and sometimes rely on certain recurring artifacts (unwanted visual patterns or flaws in images), explaining why it missed subtle frequency-domain artifacts left by generative models (9). Our model's inclusion of FFT-derived features addresses this gap by detecting patterns invisible in pixel space, making it more resilient to variations in content and post-processing. The second study that achieved 93.67% accuracy analyzed disparities in the chrominance components, the parts of an image signal that represent color information without brightness, to capture statistical differences between real images and those generated by a deep network, a type of artificial neural network (ANN) with many layers (10). This approach, known as chrominance residual analysis, detects these disparities by comparing differences between camera imaging and generative model outputs in specific color spaces (10). While it is effective at capturing color-space

Iteration	CNN-only (loss, accuracy)	CNN-FFT (loss, accuracy)
100K Train/20K Test (full dataset)	26.14%, 90.30%	17.25%, 93.30%
20K Train/4K Test (one-fifth of the full dataset)	30.47%, 88.23%	21.30%, 91.40%
10K Train/2K Test (one-tenth of the full dataset)	37.24%, 86.05%	25.12%, 89.68%
Average (loss, accuracy)	*31.28%, 88.19%*	*21.22%, 91.46%*

Table 1: Loss and accuracy for CNN-only and CNN-FFT models. Comparison of accuracy and loss for CNN-only and CNN-FFT models on different training dataset sizes. The 100K Train/20K Test label means 100,000 training images and 20,000 testing images. The 20K Train/4K Test label means 20,000 training images and 4,000 testing images. The 10K Train/2K Test label means 10,000 training images and 2,000 testing images.

	CNN-only	CNN-FFT	Percent Change
Precision	94.44%	93.20%	**-1.24%**
Recall	85.64%	93.43%	**7.79%**
F1-score	89.83%	93.31%	**3.48%**

Table 2: Performance of large CNN-only and CNN-FFT models. Comparison of precision, recall, and F1-score for CNN-only and CNN-FFT models on different training dataset sizes. The data is from the testing runs where the CNN-only model achieved a 90.30% accuracy and the CNN-FFT model achieved a 93.30% accuracy. Precision, recall and F1-score was determined based on 100,000 training images and 20,000 testing images.

https://doi.org/10.59720/24-270

statistical differences, it does not account for spatial structure or full-spectrum frequency information. Our dual-domain approach incorporates both, enabling broader adaptability across diverse generative methods and imaging conditions. The findings of the first study, which used spatial-domain CNN analysis resulting in 92.98% accuracy and the second study, which used chrominance residual analysis resulting in 93.67% accuracy, and the open questions that remain, support the need for additional research on the detection of AI-generated images to mitigate their possible negative implications. There are multiple ways to increase the strength of our integrated approach to contribute to additional research. Future work could entail implementing a user interface to automate image detection, addressing ease of use, and expanding this technology to further generative AI subtasks, including text-to-text, text-to-audio, and text-to-video. Shifting to text-to-video for a moment, using a CNN-FFT method can help detect AI-generated videos by adjusting this methodology to process video frame inputs (18). For example, OpenAI's Sora, which can create AI-generated videos, is much more accessible to the public than people might realize (19). Deepfakes, which are digitally altered images or videos, can also be created and used to spread misinformation (19). Furthermore, a larger dataset with more diverse input images and using multiple datasets to cross-validate the results from one dataset to another would give developers stronger insight into detection errors.

MATERIALS AND METHODS

A simplified pipeline of what an input image goes through is as follows: the original image is converted from the time domain to the frequency domain, then passed into the CNN for further analysis, and then image-specific features are extracted. The CNN-only model uses layers for feature extraction, dimensionality reduction, and decision-making, whereas the CNN-FFT model uses a dual-input CNN that has an add_fft_to_generator function to integrate both raw image data and their FFT representations, leveraging spatial and frequency domain features **(Figure 1)**. For both models, the

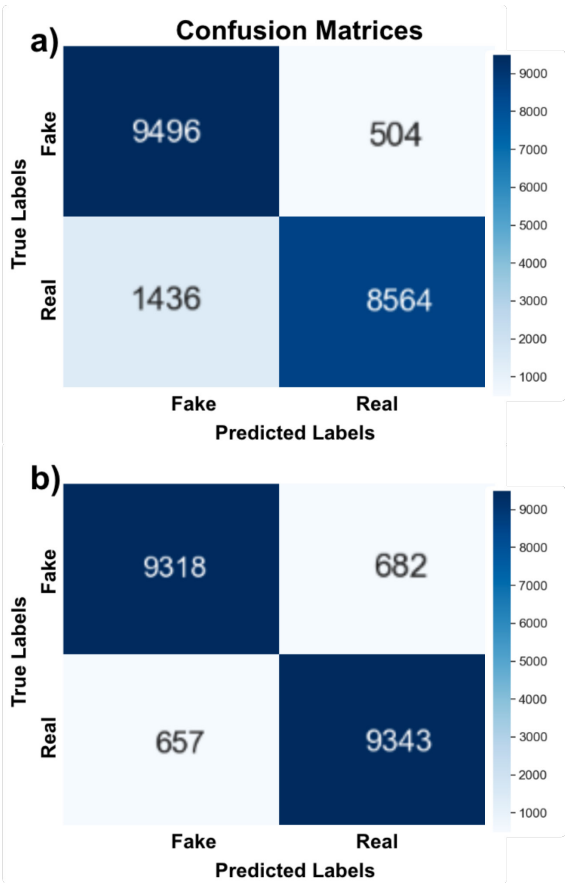

Figure 2: Confusion matrices in heatmap format of both models. Visualization of the performance of the algorithms by showing the actual versus predicted classifications using true negative (TN), false positive (FP), false negative (FN), and true positive (TP) values. TN is the top-left quadrant, FP is the top-right quadrant, FN is the bottom-left quadrant, and TP is the bottom-right quadrant. a) The confusion matrix for the CNN-only model. b) The confusion matrix for the CNN-FFT model.

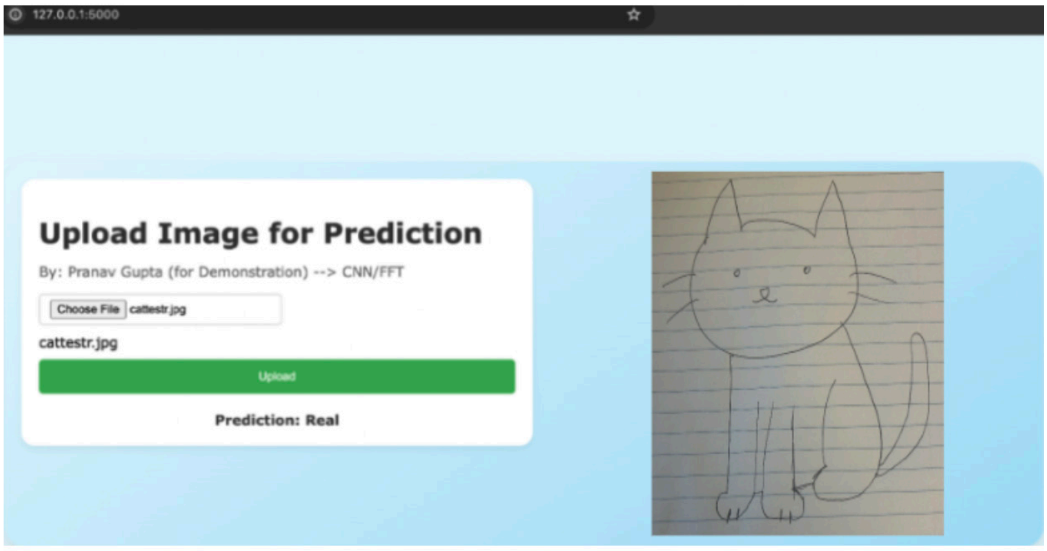

Figure 3: A user interface beta as potential future work. A user can input an image for this piece of software to detect if it is AI-generated or real. The current image is a high-level prototype as the design could change.

https://doi.org/10.59720/24-270

convolutional filter stage uses hyperparameters to function, evaluation metrics are outputted, the model is saved, and visual representations as graphical outputs are plotted.

FFT with frequency domain features

Comparing non-FFT and FFT representations of domains, information in the time domain is typically more densely populated with specific features. In contrast, information in the frequency domain is more spread out and reveals different insights. This difference results from computational processing that transforms data from the traditional (spatial) domain to the frequency domain. A Python function was implemented to perform FFT and gather the information. It transforms data from a generator by applying FFT to it, normalizes the transformed data, and yields both the original and transformed data along with the corresponding labels. It performs normalization and logarithmic scaling to make the data suitable for further processing. It also ensures the correct number of parameters corresponds to the input shape and if not, it fixes this issue by collapsing any excess parameters. Each iteration of the generator provides a batch of input data (x_batch) and labels (y_batch), and the function adds the processed FFT data to the input before yielding the normal and FFT batches for further processing or training. The mathematical formula for the logarithmic transformation is $S = \log(1 + |F(u, v)|)$. To make sure the argument of the logarithm is not zero, one is added to the magnitude. Here, $|F(u,v)|$ is the magnitude of the associated complex number, and S is the logarithmic output.

Integration of CNN and FFT

The machine learning integration for the benefits of both the CNN and FFT worked after combining the parts of the CNN architecture with features from FFT, before accessing the fully connected layers. For this study, two models were tested. The first was a model using just a CNN architecture (CNN-only). The second was a model that merged CNN architecture and information from FFT preprocessing of images (CNN-FFT).

Hyperparameter tuning

Hyperparameter tuning was done for both models, and the optimal combination of variables was chosen. Most of the changes were analogous in both models, causing FFT to be the main change being tested. The adjusted hyperparameters included the number of epochs, batch size, learning rate, and CNN architecture parameters such as dropout rate and activation functions. An epoch in machine learning is one complete pass of the training dataset through the algorithm (20). The values tested for the number of epochs were 10, 20, and 30. We used 20. Batch size is the number of training examples used in one iteration of model training (21). The values tested for batch size were 8, 16, 24, and 32. We used 32. The learning rate is a parameter within an optimization algorithm that controls the size of the steps taken during each iteration as it progresses toward the minimum of the loss function (22). The values tested for learning rate were 0.0001, 0.0002, and 0.001. We used 0.00001. The values tested for dropout rate were 0.25, 0.5, and 0.75. We used 0.5. The change in the learning rate is shown **(Figure 4a)**. This was the same in both models since the callback threshold at which the learning rate should decrease was set to be the same. This was done to reduce computational burden and optimize accuracy. In this study, the learning rate is an important hyperparameter because it is affected by the ReduceLROnPlateau callback and the binary cross-entropy loss function and is the direct parameter of the 'Adam' optimizer in the line of code that compiles the model. The different activation functions, which are also considered hyperparameters, tested were Rectified Linear Unit (ReLU) and Sigmoid. We used both in different parts of the code script.

Input and convolutional layer

The convolutional filter stage, called ‹Conv2D,› was used for the CNN-only model, while the ‹concatenate› stage was used for the CNN-FFT model. The analysis for both stages, which was done by graphing the Principal Component Analysis (PCA) of the respective layer for each of the models, is shown **(Figure 4b)**. PCA shows patterns amongst multi-dimensional data (23). Initially, inputted images went through a Conv2D layer, which was the starting point for input data. This layer is designed to handle colored (RGB) and grayscale (black and white) images with dimensions specified by img_width and img_height. The model has 32 filters, each of a 3×3 size, to scan the images. Essentially, each of these filters are small matrices that move over the image pixels to extract features such as edges, textures, or patterns, which is useful to learn from the visual data. The activation function enabled is ReLU, which introduces non-linearity into the model, allowing it to capture trends by setting all negative values to zero while preserving the positive values as they are (24). This setup contains an input_shape parameter indicating (img_width, img_height, 3), which confirms the model›s expectation of three-channel color images as input. This is the bulk of the logic of the Conv2D layer for both models. However, for the CNN-FFT model, the input_shape parameter is (img_width, img_height, 1). This is because a different breakdown of the original image is being passed in. We were able to use three as the third parameter before FFT, but this needed to be adjusted to one after FFT. The reason for this is that FFT handles single-channel data. The original three in the parameter were because RGB values were handled, which contained red, green, and blue separated values. Small matrices are referred to as kernels and coordinates, which are necessary for keeping track of what value is being scanned at one time.

Batch normalization

After the convolutional step and before the pooling layer, we found that adding batch normalization was helpful in the optimization of the model and led to an increase in positive results. Batch normalization is a method to normalize data between the layers of a neural network. This standardization does not adjust the raw data and is performed on small sections rather than the entire dataset at once. As a result, it considerably boosts the training speed and allows for higher model learning rates, thereby making the learning process more efficient and stable (25). This is done by calculating the normalized output, which can be done in two ways. The first way scales the data from a range of zero to one and the second way forces the data points to have a mean of zero and a standard deviation of one (25).

https://doi.org/10.59720/24-270

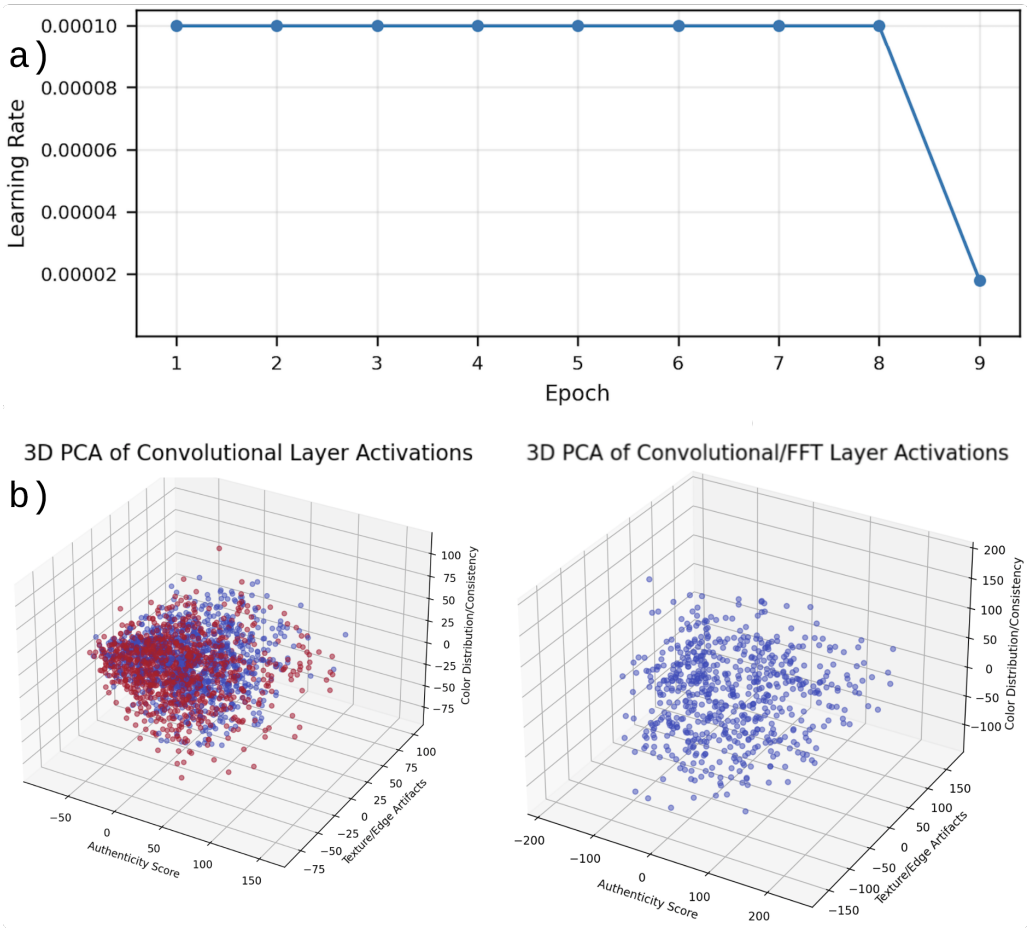

Figure 4: Learning rate over multiple epochs and PCA in 3D-graph plot format of both models. a) To enhance the performance of both models, callbacks were used to optimize training, where the drop shows the turning point where the learning rate started to decrease. The graph shown is for one iteration of training the CNN-only model. b) The layer the analysis is being done on the left side of the image is "Conv2D" for the CNN-only model and the layer that the analysis is being done on the right side of the image is "concatenate" for the CNN-FFT model.

Pooling layer

Pooling layers are typically applied after the convolutional layer. In this study, batch normalization was applied before pooling layers. Pooling can reduce the size of the input data in relation to the spatial domain. This combats potential size variance in data. Because it reduces the amount of data to be computationally run through, this helps in resource management for the hardware. Basically, the goal of pooling is to maintain the most relevant information while decreasing the input›s spatial size (26). In this study, max pooling was used as the pooling layer. This technique operates on the convolutional layers of a CNN by moving a small region, called a kernel, across the input data. It records only the largest value within each region, rather than performing matrix multiplication, which is typically used in the convolution step (27). The MaxPooling2D(2, 2) line employed in the whole CNN architecture converts the input data image from a 32×32 size to a 4×4 size, keeping important features, but having less area to analyze. This is done through three max pooling layers. Each time, the spatial size aspect is halved. The image goes from 32×32 to 16×16 (after the first max pooling layer) to 8×8 (after the second max pooling layer) to 4×4 (after the third max pooling layer). The pooling layers were followed by flattening layers that, in the context of CNNs, are used to convert a multi-dimensional array into a single-dimensional array. Each image is expressed as a multi-dimensional matrix until this point due to the layers that processed the image beforehand. Similar to how it is done in an ANN, this matrix should be converted to a single-line array of information before being connected to the fully connected layer (28).

Dropout

The CNN architecture on both models contains the dropout method with a parameter of 0.5. Dropout layers are used to decrease the amount of overfitting present in a model (29). Overfitting occurs when the model does less learning and does more memorizing of the data that is being trained on (29). Although the dataset is big, it is important to decrease this for strong accuracy and better model performance. Memorizing training data could cause the model to not understand generalizations or curveballs in test data. Dropout layers can be used to lessen this problem by randomly setting a part of the input and hidden neural network units to zero during training (29). The 0.5 parameter causes half of the units to be set to zero (30). In the CNN-FFT model, the dropout layer was applied on the fully connected layer, discussed further in the "Fully connected/Concatenated output Layer" section.

https://doi.org/10.59720/24-270

Dense layer

Two dense layers were applied to both models. A dense layer is a layer of neurons where each one uses the previous layer's neurons to work with output from the convolutional layers to classify images (30). It processes input data by receiving connections from neurons and applies a weighted sum and a non-linear activation function (30). In this study, the first dense layer contained a numerical parameter of 512 and activation of "relu". This means it has 512 neurons and uses the function that outputs the input if it is positive and zero otherwise. The second Dense layer contained a numerical parameter of one and an activation of "sigmoid". This means it has a single neuron and uses the function that maps input values to a range between zero and one, useful for binary classification (30). Dense layers are typically used towards the end of a neural network. They take in a one-dimensional input, hence the need for the flattening operation discussed earlier.

Fully connected/Concatenated output layer

The fully connected layer is the output layer of the CNN's architecture, which means that all neurons in the previous layer are connected to the subsequent layer. The output of the preceding layers affects the outcome after the last layer (31). In the CNN model, the dense layer is this layer. A fully connected layer utilizes patterns of image recognition gathered throughout training the model on the dataset. The output layer, as demonstrated by the Dense layer, gives us our classification for the image. The activation function is "sigmoid", which outputs a value between zero and one. It converts the input, which can be any value from negative infinity to positive infinity, into a probability score that can then be interpreted into a certain class, such as AI-generated or real. There are two separate blocks of CNN architecture in this model **(Figure 5a)**. The first did the steps described in the earlier subsections on the image without FFT and saved this to a local variable (32, 33). The second did similar steps on the image with FFT and saved this to another local variable. Features were then saved by combining the information gathered from each block of CNN architecture by using the concatenate method **(Figure 5b)**. This lets the model harness information from both approaches, helping it use this more for its classification (31-33). The combined information was then turned into a fully connected layer following similar steps as the CNN-only model (34).

Confusion matrices

Confusion matrices typically show information about the four types of scores. This can help determine how successful a classification model is, in addition to other features such as accuracy, loss, recall, precision, and F1-score.

TP: objects of a certain class that are correctly recognized in that class.

TN: objects not belonging to a certain class that are not recognized in that class.

FP: objects not belonging to a certain class that are incorrectly recognized in that class.

FN: objects of a certain class that are not recognized in that class.

The general equations for the aforementioned additional features (besides loss since that depends on the specific loss function being used) involving these four scores are as follows (16):

$$Accuracy = \frac{correct\ classifications}{total\ classification} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

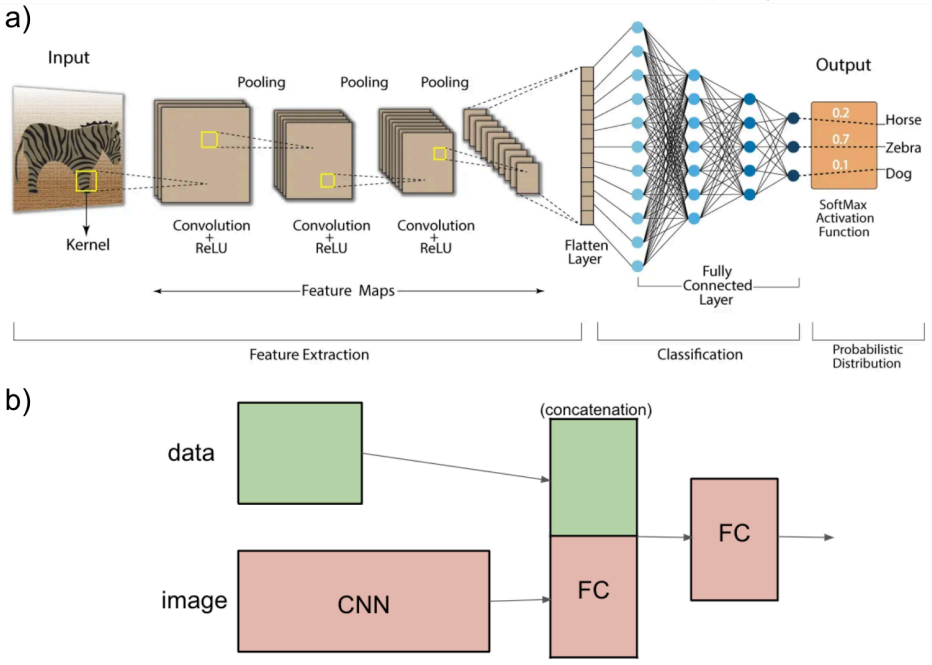

Figure 5: Visuals of a fully connected layer in a CNN and the concatenation between CNN and FFT. a) An input image goes through multiple layers before reaching the fully connected layer. b) Information from the image processed through the CNN is combined with extracted data from the frequency domain. This is then sent to the final fully connected layer as one entity passed in.

https://doi.org/10.59720/24-270

Training/Testing dataset and Python usage

Both models were tested on the same comprehensive dataset. This dataset was the "CIFAKE: Real and AI-Generated Synthetic Images" by Jordan J. Bird from the platform called Kaggle (35). This dataset contained a total of 120,000 images. Each image was 32×32 and had colored and grayscale aspects. The split was 100,000 training images and 20,000 testing images. Training images were split into 50,000 AI-generated images and 50,000 real images. Similarly, testing images were split into 10,000 AI-generated images and 10,000 real images. During development, we created two smaller datasets that stemmed from this larger dataset, which saved computational resources by testing on a smaller dataset when needed. One of the smaller datasets contained 24,000 (one-fifth) of the total images and the other smaller dataset contained 12,000 (one-tenth) of the total images. We wrote a Python script to gather these images randomly. The ratio of images for training and testing remained the same in all datasets.

The version of Python used is 3.8.18 (36). The version of the TensorFlow library used is v2.13.0 (37). The version of the NumPy library used is v1.22.3 (38). The version of the Matplotlib library used is v3.7.5, and mpl_toolkits.mplot3d was used from the Matplotlib Toolkit (39). The version of the Scikit-learn library used is v1.3.2 (40). The version of the Seaborn library used is v0.13.2 (41). The OS Module used was from the Python Standard Library (42). The Shutil Module used was from Python Standard Library (43). The Random Module used was from the Python Standard Library (44). Towards the end of the study, because of the lengthy times for training the model, after a few iterations of testing, both models were saved as a .h5 file to be loaded in and used whenever there was no need for re-training.

CPU drawback

A challenge we encountered was testing on CPU and as a result, we pivoted to testing on GPU using a virtual environment to improve the time and computing power used. In fact, the CNN-FFT model took more than double the time to finish training. This was because more happened with each input image directly from the dataset. As discussed, each image had to be preprocessed through FFT and then input into the CNN architecture to determine its classification. If only the CNN was used in the model, operations performed used fewer system resources, causing it to complete each epoch faster. The hardware used was an Apple Silicon (M1 chip) MacBook Air with an 8-core CPU and 8-core GPU. The project code was written in the Python language using TensorFlow optimized for the ARM64 computer system architecture and programmed in the Visual Studio Code application (v1.87.0) (45).

Received: July 22, 2024
Accepted: January 18, 2025
Published: October 24, 2025

REFERENCES

1. Wu, Yonghui, and David Fleet. "How AI Creates Photorealistic Images from Text." *Google Research Blog*, 22 Jun. 2022, https://www.blog.google/technology/research/how-ai-creates-photorealistic-images-from-text/. Accessed 14 Mar. 2025.
2. Valyaeva, Alina. "AI Image Statistics: How Much Content Was Created by AI." *Everypixel Journal*, 15 Aug. 2023, https://www.journal.everypixel.com/ai-image-statistics. Accessed 3 Apr. 2024.
3. Fortino, Andres. "Embracing Creativity: How AI Can Enhance the Creative Process." *SPS NYU Emerging Technologies Collaborative Blog*, 2 Nov. 2023, https://www.sps.nyu.edu/about/news-and-ideas/articles/etc/2024/embracing-creativity-how-ai-can-enhance-the-creative-process.html. Accessed 15 Jun. 2025.
4. Sorkin, Andrew Ross, et al. "AI-Enhanced Fake Pentagon Explosion Ripples Through Stock Market." *The New York Times*, 23 May 2023, https://www.nytimes.com/2023/05/23/business/ai-picture-stock-market.html. Accessed 10 Apr. 2024.
5. Wallace, Danielle. "Fake Pentagon Explosion Image Goes Viral on Twitter, Sparking Further AI Concerns." *Fox Business*, 23 May 2023, https://www.foxbusiness.com/technology/fake-pentagon-explosion-image-goes-viral-twitter-sparking-further-ai-concerns. Accessed 10 Apr. 2024.
6. Zhu, Mingjian, et al. "GenImage: A Million-Scale Benchmark for Detecting AI-Generated Image." *arXiv*, 24 Jun. 2023, https://doi.org/10.48550/arXiv.2306.08571.
7. "What Are Convolutional Neural Networks?" *IBM*, 4 Jun. 2025, https://www.ibm.com/think/topics/convolutional-neural-networks. Accessed 16 Jun. 2025.
8. "Convolutional Neural Networks (CNNs / ConvNets): Pooling Layer." *CS231n: Deep Learning for Computer Vision*, https://www.cs231n.github.io/convolutional-networks/#pool. Accessed 10 Aug. 2025.
9. Bird, Jordan J., and Ahmad Lotfi. "CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images." *arXiv*, 24 Mar. 2023, https://doi.org/10.48550/arXiv.2303.14126.
10. Li, Haodong, et al. "Detection of Deep Network Generated Images Using Disparities in Color Components." *Signal Processing*, vol. 174, no. 107616, Sep. 2020, https://doi.org/10.1016/j.sigpro.2020.107616.
11. Epstein, David C, et al. "Online Detection of AI-Generated Images." *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 25 Dec. 2023, pp. 382-392. https://doi.org/10.1109/ICCVW60793.2023.00045.
12. Nair, Varsha, et al. "Fast Fourier Transformation for Optimizing Convolutional Neural Networks in Object Recognition." *arXiv*, 8 Oct. 2020, https://doi.org/10.48550/arXiv.2010.04257.
13. Yan, Shilin, et al. "A Sanity Check for AI-Generated Image Detection." *arXiv*, 27 Jun. 2024, https://doi.org/10.48550/arXiv.2406.19435.
14. Durall, Ricard, et al. "Watch Your Up-Convolution: CNN-Based Generative Deep Neural Networks Are Failing to Reproduce Spectral Distributions." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 5 Aug. 2020, pp. 7887-7896. https://doi.org/10.1109/CVPR42600.2020.00791.
15. Holdsworth, Jim and Mark Scapicchio. "What Is Deep Learning?" *IBM*, https://www.ibm.com/think/topics/deep-learning. Accessed 5 Apr. 2024.

https://doi.org/10.59720/24-270

16. "Classification: Accuracy, Recall, Precision, and Related Metrics | Machine Learning | Google for Developers." *Google Developers*, https://www.developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall. Accessed 14 Mar. 2025.
17. Donovan, Tom. "What Is N log N?" *Built In*, https://www.builtin.com/software-engineering-perspectives/nlogn. Accessed 14 Mar. 2025.
18. Chiariglione, Leonardo. "What Is the Difference between an Image and a Video Frame?" *Leonardo's Blog*, 17 Oct. 2019, https://www.blog.chiariglione.org/what-is-the-difference-between-an-image-and-a-video-frame/. Accessed 16 Apr. 2024.
19. Cerullo, Megan. "OpenAI's New Text-to-Video Tool, Sora, Has One Artificial Intelligence Expert 'Terrified.'" *CBS News*, CBS Interactive, https://www.cbsnews.com/news/openai-sora-text-to-video-tool/. Accessed 22 Mar. 2025.
20. "What Is Epoch in Machine Learning?: UNext." *UNext*, 7 Mar. 2023, https://www.u-next.com/blogs/machine-learning/epoch-in-machine-learning/. Accessed 14 Apr. 2024.
21. "Batch Size in Deep Learning." *Ultralytics*, 14 Jun. 2025, https://www.ultralytics.com/glossary/batch-size. Accessed 18 Jun. 2025.
22. Belcic, Ivan and Cole Stryker. "What Is Learning Rate in Machine Learning?" *IBM*, 17 Apr. 2025, https://www.ibm.com/think/topics/learning-rate. Accessed 19 Jun. 2025.
23. Lever, Jake, et al. "Principal Component Analysis." *Nature Methods*, vol. 14, 29 Jun. 2017, pp. 641-642. https://doi.org/10.1038/nmeth.4346.
24. Riva, Martin. "Batch Normalization in Convolutional Neural Networks." *Baeldung on Computer Science*, 18 Mar. 2024, https://www.baeldung.com/cs/batch-normalization-cnn. Accessed 8 Apr. 2024.
25. Chaudhary, Kartik. "Understanding Audio Data, Fourier Transform, FFT, Spectrogram and Speech Recognition." *Medium* (Towards Data Science), 4 Jun. 2021, https://www.medium.com/data-science/understanding-audio-data-fourier-transform-fft-spectrogram-and-speech-recognition-a4072d228520. Accessed 4 Apr. 2024.
26. "Pooling Layers in CNN." *Deepchecks*, 11 Apr. 2023, https://www.deepchecks.com/glossary/pooling-layers-in-cnn/. Accessed 8 Apr. 2024.
27. "Max Pooling." *DeepAI*, 17 May 2019, https://www.deepai.org/machine-learning-glossary-and-terms/max-pooling. Accessed 8 Apr. 2024.
28. "Do Convolutions 'Flatten Images'?" *GeeksforGeeks*, 10 Feb. 2024, https://www.geeksforgeeks.org/category/ai-ml-ds/page/183/?type=recent. Accessed 8 Apr. 2024.
29. "Dropout in Neural Networks." *Dremio*, https://www.dremio.com/wiki/dropout-in-neural-networks/. Accessed 9 Apr. 2024.
30. Dumane, Govinda. "Introduction to Convolutional Neural Network (CNN) Using Tensorflow." *Medium* (Towards Data Science), 19 Mar. 2020, https://www.medium.com/data-science/introduction-to-convolutional-neural-network-cnn-de73f69c5b83. Accessed 10 Apr. 2024.
31. Gombru, Raúl. "Concatenate Layer Output with Additional Input Data." *PyTorch Forums*, 29 Jun. 2018, https://www.discuss.pytorch.org/t/concatenate-layer-output-with-additional-input-data/20462. Accessed 14 Apr. 2024.
32. Haque, Kh N. "What Is Convolutional Neural Network – CNN (Deep Learning)." *LinkedIn*, 3 Apr. 2023, https://www.linkedin.com/pulse/what-convolutional-neural-network-cnn-deep-learning-nafiz-shahriar/. Accessed 13 Apr. 2024.
33. E, Swapna K. "Convolutional Neural Network: Deep Learning." *Developers Breach*, 2022, https://www.developersbreach.com/convolution-neural-network-deep-learning/. Accessed 11 Aug. 2024.
34. "Home En." *FFT – NTI-Audio*, https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft. Accessed 7 Apr. 2024.
35. Bird, Jordan J. "CIFAKE: Real and AI-Generated Synthetic Images." *Kaggle*, 28 Mar. 2023, https://www.kaggle.com/datasets/birdy654/cifake-real-and-ai-generated-synthetic-images. Accessed 26 Feb. 2024.
36. "Python Release Python 3.8.10." *Python,* https://www.python.org/downloads/release/python-3810/. Accessed 14 Jun. 2024.
37. "API Documentation v2.13." *TensorFlow,* https://www.tensorflow.org/versions/r2.13/api_docs. Accessed 20 Apr. 2024.
38. Harris, Charles R, et al. "Array Programming with NumPy." *Nature*, vol. 585, Sep. 2020, pp. 357-362. https://doi.org/10.1038/s41586-020-2649-2.
39. Hunter, John D. "Matplotlib: A 2D Graphics Environment." *Computing in Science & Engineering*, vol. 9, no. 3, 18 Jun. 2007, pp. 90-95. https://doi.org/10.1109/MCSE.2007.55.
40. "1. Supervised Learning." *Scikit-learn*, https://www.scikit-learn.org/stable/supervised_learning.html. Accessed 19 Jun. 2025.
41. Waskom, Michael L. "Seaborn: Statistical Data Visualization." *Journal of Open Source Software*, vol. 6, no. 60, 6 Apr. 2021, pp. 3021-3024. https://doi.org/10.21105/joss.03021.
42. "OS Module." *Python Documentation*, https://www.docs.python.org/3/library/os.html. Accessed 20 Apr. 2024.
43. "Shutil Module." *Documentation*, https://www.docs.python.org/3/library/shutil.html. Accessed 20 Apr. 2024.
44. "Random Module." *Documentation*, https://www.docs.python.org/3/library/random.html. Accessed 20 Apr. 2024.
45. "Updates (version 1.87)." *Documentation*, https://code.visualstudio.com/updates/v1_87. Accessed 10 Aug. 2025.