Detecting Images Generated by Popular Deep Learning Models

Course: INFO-H518 Deep Learning
Team: Sundeep Kakar and Maurice Reeves

1. Introduction

Artificial images generated by deep learning models have revolutionized the field of computer vision and creativity. These images are produced by complex algorithms instead of traditional methods, opening up a world of possibilities. The journey of artificial image generation began in 2014 with the introduction of Generative Adversarial Networks (GANs) by Ian Goodfellow and his team. By 2018, synthetic image generation had gained mainstream popularity. Since then, numerous companies have developed free Artificial General Intelligence (AGI) models specifically tailored for generating photos. Examples include DALL-E by OpenAI, which creates images based on textual descriptions, Midjourney by MidJourney Inc, known for its ability to generate high-quality artistic images, and Stable Diffusion by Stability AI, which specializes in creating stable and diverse outputs. With further research and development, these technologies will continue to push the boundaries of creativity and innovation in the years to come.

Image generated using MidJourney

2. Problem

With the advent of such models, a new host of problems has emerged. At first, early models such as DALL-E would churn out images that were visibly fake but now the latest models are creating images that are very lifelike, devoid of imperfections, and hard for humans to distinguish between. Numerous negative applications of such technology include but are not limited to, identity theft, fake advertisement campaigns, and critically aimed disinformation. Preventing bad actors from using such technologies can be hard given the non-existent barrier to entry to such technologies. Some models are open-source while most do not require identification to use.

A hypothesis can be made that if an image is AI-generated, it could include key patterns or digital fingerprints. If so, Deep Learning models such as Convolutional Neural Networks, should be able to detect and classify such patterns.

3. Project scope

The project scope comprised of building and researching rich data sources that contain real and fake i.e. non-AI generated vs AI-generated images that we could train deep learning models on. The project was conducted in 2 parts. Part 1 focused on training the model on a dataset of a given subject, in our case, shoes. Part 2 focused on training a much more complex model on a generalized dataset.

4. Methodology: Part 1 - Shoes

We built a dataset of 2000 images comprising real and AI-generated images of shoes. This was built by our team and hosted on Kaggle. This was built by sourcing 3 separate Kaggle Datasets: Real images were scraped from Google Images for Nike, Adidas, and Converse shoes. The source Kaggle dataset is linked below under references and the AI-Generated images were generated using MidJourney.

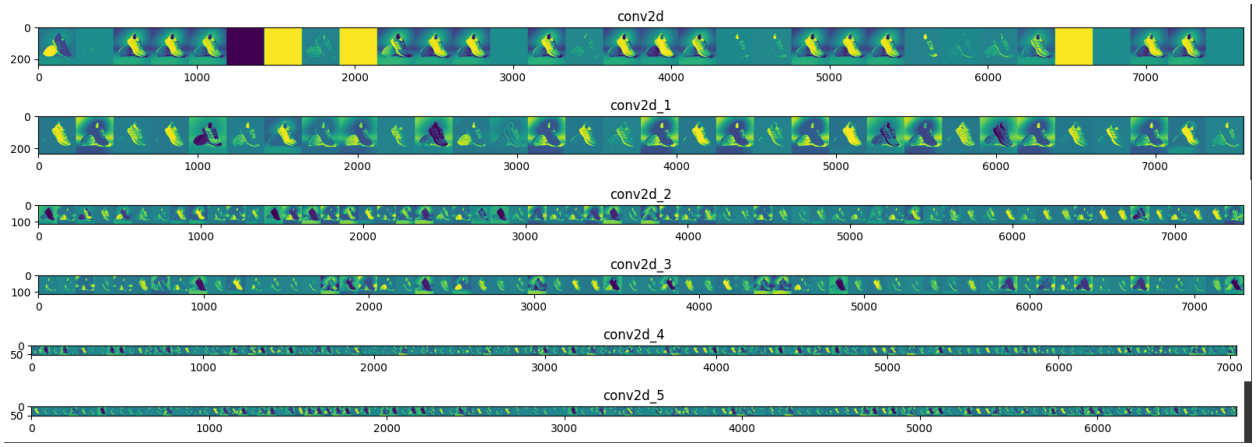
Examples of shoe images from the dataset: AI (left) vs Real (right)

Convolutional Neural Network trained on this dataset to produce the highest test accuracy of ~95% (see Appendix A). The model had high accuracy with only 4 false negatives and 70 false positives out of 2000 images. Analyzing the false negative images revealed patterns at first glance. The filters for each convolutional layer are two-dimensional filters that can be examined and depicted to unveil the varieties of features it detects. Moreover, delving into the activation maps generated by convolutional layers allows a precise understanding of the features identified for a specific input image.

Examples of false-positives and false-negatives

Visualizing these filters involved iterating through only the convolutional layers and extracting weights and biases for each layer. The weights were then normalized and plotted for each channel, in our case, RGB. After visualizing multiple filters for true-positive (AI) and false-positive (real classified as AI)

images, a few patterns stood out. The filters are easy to comprehend in the first steps and then become hard to interpret visually for humans. However, as the filters progress from layer to layer, we can see which areas of interest were taken into account as the brighter regions.



False-positive (real images classified as AI-generated) were used to visualize filters.

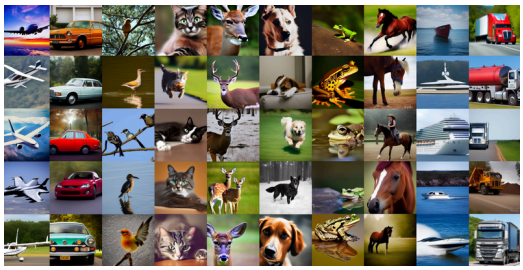
A few key commonalities in false-positives were:

- Image effects: vignettes and angled-lighting on the shoes
- Shape: relative size of the shoe to the image, orientation of the shoes (stacked, non-sideways) and placement of the second shoe being different than the first
- Background: artifacts in the background (which was more common with AI images), detailed backgrounds.
- Colors: diversity of colors/patterns in the shoes (more common for AI-generated shoes).

Some of these are visible in the false positive images above. This process allowed us to make inferences on the data using a focused dataset with larger images (240x240).

5. Methodology: Part 2 - CIFAKE

The dataset titled “CIFAKE” is a dataset that contains 60,000 synthetically generated images and 60,000 real images (collected from CIFAR-10). The images are 32x32 RGB images and the model used to generate the AI counterpart is Stable Diffusion version 1.4. In the dataset, there are 100,000 images for training (50,000 per class) and 20,000 for testing (10,000 per class). 70% was used for training, 20% for validation, and 10% for testing.

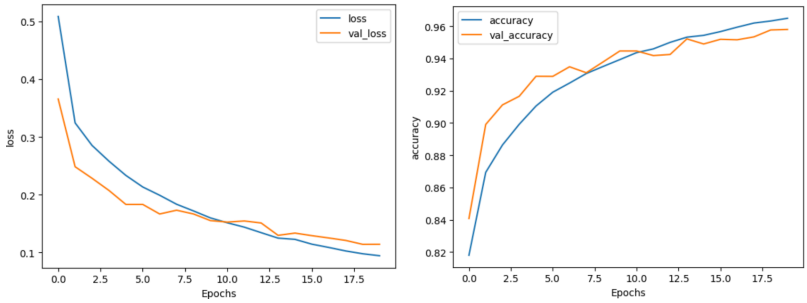


Example of images in the CIFAKE dataset

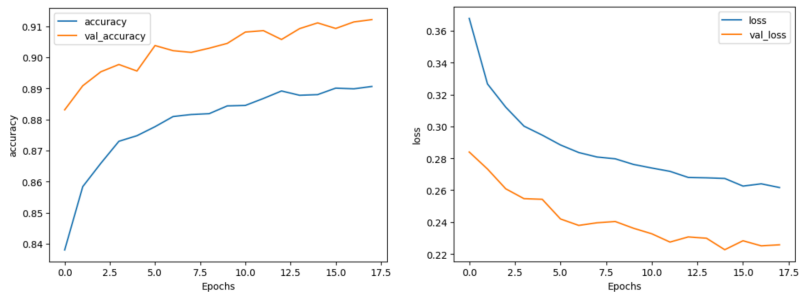
Infrastructure: Tensorflow and Keras were used to build all models since Keras is easier to use and debug when working with complex models all while handling GPU management smoothly. The training was carried over 2 platforms: Google Colab using NVIDIA GPU L4 and Kaggle using NVIDIA GPU P100.

Data Preprocessing: Using data augmentation techniques such as horizontal flips, vertical flips, zoom, and brightness adjustment new data was augmented from existing data. This led to increased accuracy and reduced overfitting in the final model while iterating. The image data was parsed using ImageDataLoader from Keras.

- *Custom CNN*: Comprising of 3 [ConvL, BachNorm, ConvL, BatchNorm, MaxPool] sections attached in a link which is flattened and followed by 3 Dense layers feeding a sigmoid final activation function. All layers use relu as the activation function. During testing, the model showed a minimum loss of 0.0945 and a maximum accuracy of 95.81%.

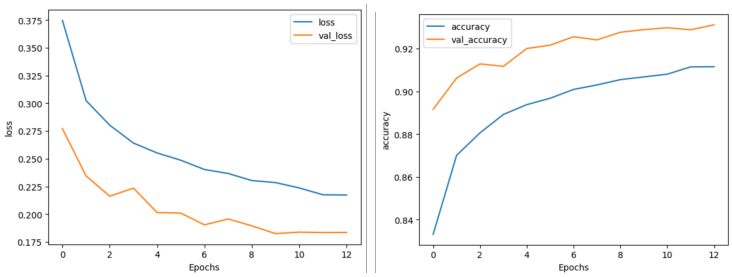


- *MobileNet50V2*: Mobilenet50V2 is a lightweight, efficient solution with 3.5M parameters. Fine-tuning a model with Resnet50V2 and custom top to classify the images was a good pre-trained model approach given that its lightweight, has good performance, and has a smaller-than-average number of parameters. Furthermore, new fully-connected layers on top of the model can be added, however, a single layer led to very good performance. This model takes up more space and requires more parameters than MobileNet50V2 directly. The model must demonstrate a significant enhancement in performance compared to the MobileNet model to justify its adoption, considering the latter's lightweight nature. After testing, this model showed a minimum loss of 0.2227 and a maximum test accuracy of 91.22%.



- *ResNet50V2*: ResNet50V2 is a lightweight, efficient solution with 25.6M parameters. ResNet50V2 is a modified version of ResNet50 that performs better than ResNet50 and ResNet101 on the ImageNet dataset. It defers to our previous models such that its architecture contains 50 Residual CNN layers. This model had a minimum test loss of 0.1873 and a maximum test accuracy of 92.83%. To avoid overfitting in the models a larger dropout rate of 0.5 was selected. This, while large, allowed for the model's loss graph to be consistent across epochs

without large abrupt changes. Compared to MobileNet50V2, ResNet50V2 had a shorter graph between the training-test loss graphs and the accuracy graphs.



Hyperparameters: Hyperparameters were tweaked in order to find the best fit for our accuracy. Multiple optimizers such as RMSprop and SGD were tested but Adam resulted in the most accurate solution. The learning rate was also tested across the spectrum and a learning rate of 0.0001 was optimal over the default learning rate of 0.001. The high dropout rate is evident in the model performance graphs above for ResNet50V2 and MobileNet50V2 however this offered protection against inconsistent training and overfitting in the final steps of the process. All models discussed were iterated multiple times over a variation of hyperparameters to achieve the best accuracy.

This process resulted in the following table:

Model	**Parameters (in M)**	**Minimum Test Loss**	**Maximum Test Accuracy (%)**	**Training time under similar GPU conditions (in ms/step)**
Custom CNN	59	0.0945	95.81	295
MobileNet50V2	3.5	0.2227	91.22	225
Resnet50V2	25.6	0.1873	92.83	223

From the table, it is visible that Custom CNN has the best overall performance with close to 96% accuracy across the large dataset. The model achieves impressive accuracy in predicting AI-generated images, demonstrating its strong proficiency in distinguishing between AI and real images. This notable level of accuracy positions the model as a dependable solution for tasks such as detecting potentially deceptive or harmful AI-generated content across a spectrum of applications.

6. Conclusion

In summary, the rise of sophisticated deep learning models presents new challenges, as they produce increasingly realistic images that can be exploited for nefarious purposes like identity theft and disinformation campaigns. Addressing this issue requires detecting unique patterns in AI-generated images through Convolutional Neural Networks and other Deep Learning solutions. By developing effective detection methods, we can mitigate the risks associated with the misuse of this technology and protect all against its harmful effects.

7. References

Dertat, Arden. "Applied Deep Learning — Part 4: Convolutional Neural Networks." Towards Data Science, 2022, https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2#9722. (Accessed: 01 May 2024).

Thompson, S.A. (2024) Test yourself: Which faces were made by A.I.?, The New York Times. Available at: https://www.nytimes.com/interactive/2024/01/19/technology/artificial-intelligence-image-generators-faces-quiz.html (Accessed: 01 May 2024).

Dertat, A. (2017) Applied deep learning - part 4: Convolutional Neural Networks, Medium. Available at: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2#9722 (Accessed: 01 May 2024).

He, K. et al. (2015) Deep residual learning for image recognition, arXiv.org. Available at: https://arxiv.org/abs/1512.03385 (Accessed: 01 May 2024).

8. Appendix - A

- CNN Model for Shoe Detection:

```
# Define a CNN model using Keras
model = tf.keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(240, 240, 3)),  # 238 x 238 x 32
    layers.BatchNormalization(),
    layers.Conv2D(32, (3, 3), activation='relu'),  # 236 x 236 x 32
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),  # 118 x 118 x 32
    layers.Conv2D(64, (3, 3), activation='relu'),  # 116 x 116 x 64
    layers.BatchNormalization(),
    layers.Conv2D(64, (3, 3), activation='relu'),  # 114 x 114 x 64
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),  # 57 x 57 x 64
    layers.Conv2D(128, (3, 3), activation='relu'),  # 55 x 55 x 128
    layers.BatchNormalization(),
    layers.Conv2D(128, (3, 3), activation='relu'),  # 53 x 53 x 128
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),  # 26 x 26 x 128
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.25),
    layers.Dense(256, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

- Custom CNN Model for CIFAKE Classification:

```
# Define a CNN model using Keras
model = tf.keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)),  # 238 x 238 x 32
    layers.BatchNormalization(),
    layers.Conv2D(32, (3, 3), activation='relu'),  # 236 x 236 x 32
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),  # 118 x 118 x 32
    layers.Conv2D(64, (3, 3), activation='relu'),  # 116 x 116 x 64
    layers.BatchNormalization(),
    layers.Conv2D(64, (3, 3), activation='relu'),  # 114 x 114 x 64
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),  # 57 x 57 x 64
    layers.Conv2D(128, (3, 3), activation='relu'),  # 55 x 55 x 128
    layers.BatchNormalization(),
    layers.Conv2D(128, (3, 3), activation='relu'),  # 53 x 53 x 128
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),  # 26 x 26 x 128
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.25),
    layers.Dense(256, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```