AI Image Classifier: Learning to Identify AI-Generated Images & Faces using Binary Image Classification

Antonio Villarreal, Taise Miyazumi, Tyler Wong, Joshua Kim, Daniel Franco

Abstract

With the advent of artificial intelligence tools such as ChatGPT and DallE 3 there has been an ever growing concern with the implications of this new technology. Our team has successfully trained two convolutional neural networks, one to detect AI generated faces and another for general AI generated images. Each CNN has five common convolutional blocks consisting of batch normalization and max pooling. These convolutional blocks are stacked together to form the overall architecture of our CNN. After training our two CNNs, we were able to achieve a validation accuracy of 98.08% and 93.56%. While it may be difficult for individuals to differentiate real images generated from AI generated ones, we can harness the power of machine learning to do the heavy lifting for us. As these technologies advance, it's imperative that we as a society understand the risks of such powerful technology and find methods to collaborate with corporations/governments to ensure public safety and wellbeing.

1. Introduction

The increasing sophistication of face image generation technology has caused an unprecedented threat to our trust in digital media, the protection of individual identities, and the prevention of misinformation. As advanced generative machine learning models like DallE and ChatGPT produce increasingly realistic outputs, distinguishing AI-generated content from real content has become an increasingly daunting task. Our team plans to develop a convolutional neural network

(CNN) that can accurately classify images as either real or AI-generated. We intend to train the model using a labeled dataset from Kaggle. This model will support an application, deployed using Streamlit and Flask, that allows users to upload images to determine their origin. Through this project, we aim to equip users with the necessary tools to discern between AI-images and man made images.

2. Related Work

Binary Image Classification involves using algorithms or models to categorize images into one of two distinct groups, in this case real or machine generated images. Typically deep learning models are trained for this task due to their complex structure allowing for more detailed feature learning and expression [1].

Convolutional Neural Networks (CNNs) are particularly effective for this task due to their complex structures that excel in feature learning and expression. These networks utilize convolutional layers for extracting important visual features and pooling layers to reduce data dimensionality, enhancing processing efficiency and feature detection robustness.
CNNs are trained on large datasets with deep learning techniques, improving their accuracy in classifying new images. They outperform traditional machine learning methods by better handling the intricacies of visual data in large-scale identification tasks in computer vision [2].

3. Methodology

3.1 Datasets

Since we opted for a fully supervised framework, our datasets consist entirely of labeled images. This goes for both of the datasets used, one consisting of a broad range of different images while the other consists entirely of human faces. The dimensions of these images vary but all images are resized to a resolution of 224x224 during training.

3.1.1 General Images

For the image classifier, we employed a dataset of 174,427 images [3]. Of these images, 91,993 are real images. They can be anything from authentic photographs of landscapes, cities, animals, or portraits. As shown in Figure 1, these images display a wide variety of different things.

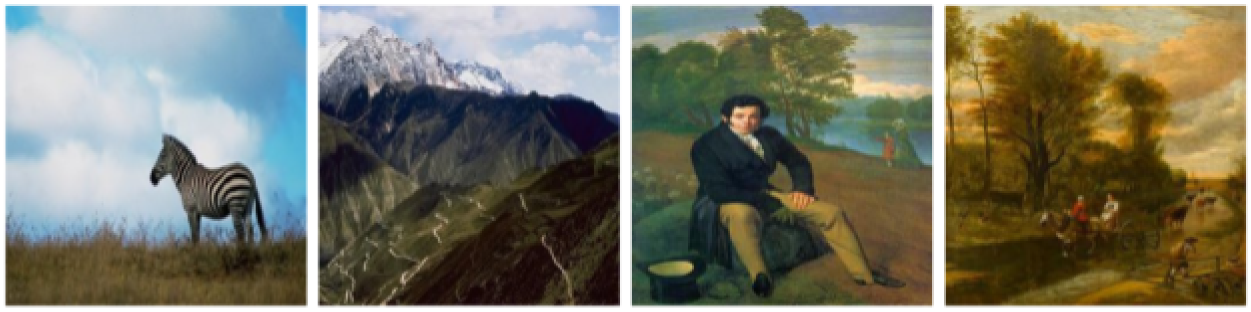


Figure 1 - Showcasing the variety of real images

The rest of the dataset is comprised of 82,434 AI artistic renderings. Once again, the contents of these images vary immensely. Just like the real image subset, the AI generated subset contains drawings, portraits, and landscapes. Though some of these images bear striking features indicative of an AI generated image, others might prove more difficult to identify as real or fake to the naked eye. In Figure 2, the first image can easily be distinguished by its awkward positioning of the boat while the second displays a poor attempt at drawing human hands while the other two images on the right can easily be mistaken as authentic drawing

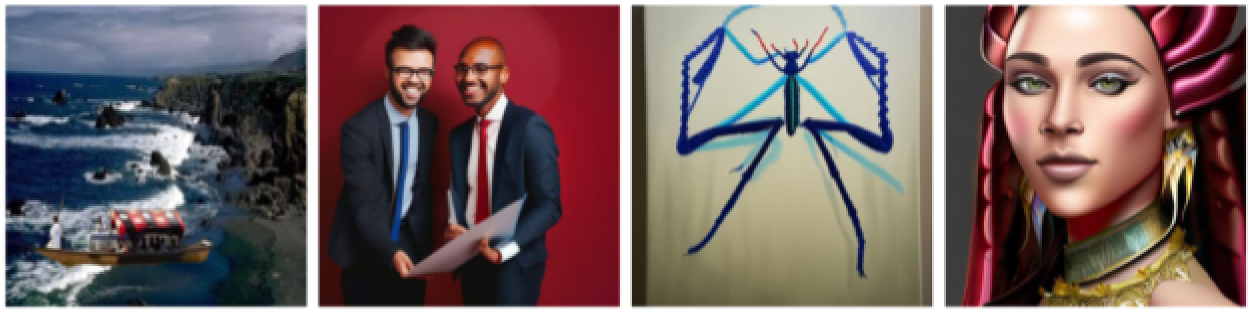


Figure 2 - Not all AI generated images are as easy to identify as others

3.1.2 Human Faces

The face classifier follows a similar pattern as the image classifier. The dataset for this section consists of 140,000 images and is split evenly with 70,000 real human faces collected from Flickr and another 70,000 fake images generated by StyleGAN [4]. Additionally, each subset is further split three ways with a roughly 71% split for training and 14% for both testing and validation. This comes out to a total of 100,000 training images, 20,000 for testing and 20,000 for validation. At first glance, it might appear that all the images in Figure 3 are taken from real photos but the two images on the left are AI generated while the two on the right are real. The striking realism of these AI images is what led us to this project.

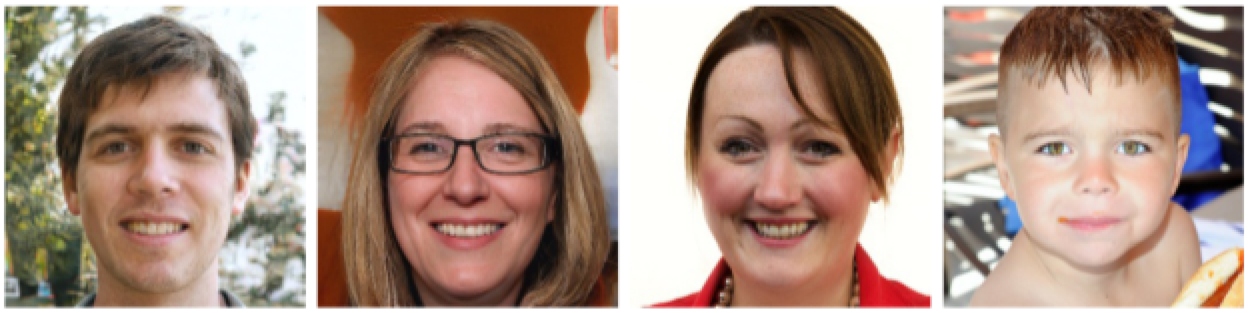


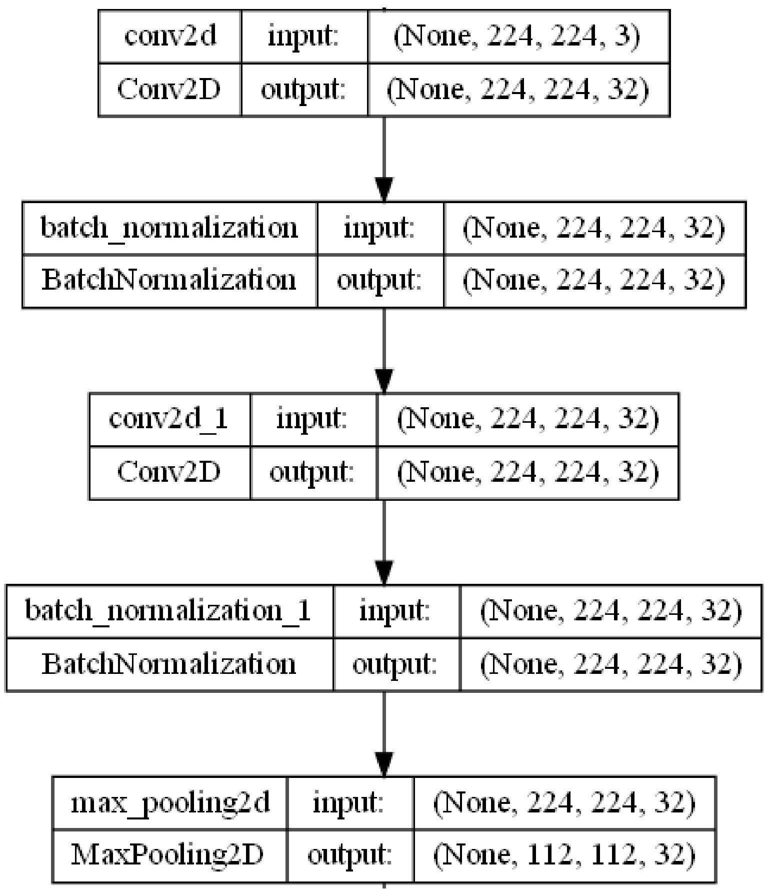
Figure 3 - It is surprisingly difficult to identify the StyleGAN images (two on left)

3.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a widely used architecture for image classification tasks due to their ability to capture spatial hierarchies in data effectively and to be highly customizable towards the specific use case. Below the CNN architecture is designed to accept input of 224 x 224 images and detect if the input image is AI-generated or not. Specifically, two CNNs were trained, one to detect AI-generated faces and one to detect AI-generated images.

3.2.1 Model Components

SINGLE CONVOLUTIONAL BLOCK (5)



The model architecture consists of five common convolutional blocks, each comprising a convolutional layer followed by batch normalization and max pooling. These convolutional blocks are stacked to form the overall architecture.

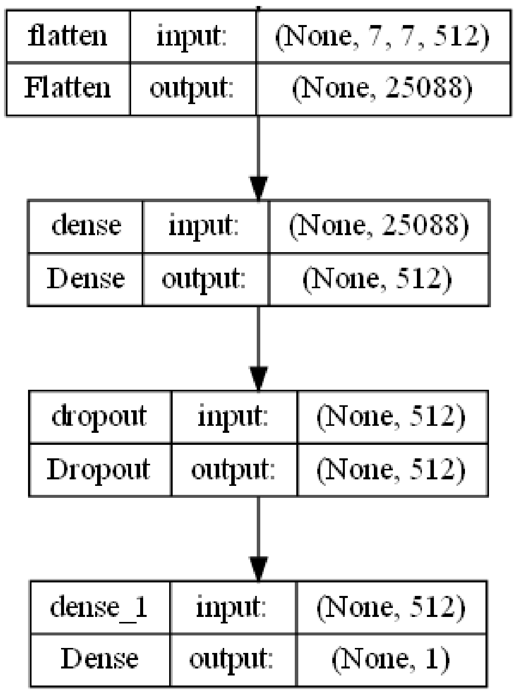
Convolutional Layers
The convolutional layers in the network are responsible for extracting features from the input images. Each convolutional layer consists of multiple filters that convolve over the input data to

detect various patterns and features. The use of rectified linear unit (ReLU) activation function promotes non-linearity, enabling the model to learn complex relationships within the data.

<u>Batch Normalization</u>
Batch normalization is incorporated after each convolutional layer to standardize the inputs to the activation function, which helps in stabilizing and accelerating the training process. It normalizes the activations of each layer, reducing internal covariate shifts and allowing for faster convergence during training.

<u>2D Max Pooling</u>
Max pooling layers are employed to downsample the feature maps, reducing the spatial dimensions of the data while retaining the most relevant information. This helps in making the model more computationally efficient and robust to variations in input data.

FINAL CONVOLUTIONAL BLOCK (1)



flatten | input: | (None, 7, 7, 512)
Flatten | output: | (None, 25088)
dense | input: | (None, 25088)
Dense | output: | (None, 512)
dropout | input: | (None, 512)
Dropout | output: | (None, 512)
dense_1 | input: | (None, 512)
Dense | output: | (None, 1)

The final block of the model architecture consists of a flattening layer followed by two dense layers. The flattening layer converts the multidimensional output of the preceding convolutional layers into a one-dimensional array, which is then fed into dense layers for classification, with a dropout layer for regularization.

<u>Dense Layers</u>
Dense layers serve as the high-level reasoning component of the network, synthesizing the hierarchical features extracted by the convolutional layers into a decision regarding the

classification of the input image. Two key aspects of the dense layers are the choice of activation functions: Rectified Linear Unit (ReLU) and Sigmoid.

ReLU Activation Function

The ReLU activation function, employed in the hidden layers of the dense network, introduces non-linearity to the model. This nonlinearity is crucial for enabling the network to learn complex relationships and patterns within the data. ReLU has several advantages, including computational efficiency and avoidance of the vanishing gradient problem, which can occur with other activation functions like sigmoid or tanh. By allowing only positive values to pass through, ReLU ensures that the network can capture complex features without being constrained by saturation issues encountered in traditional activation functions.

Sigmoid Activation Function

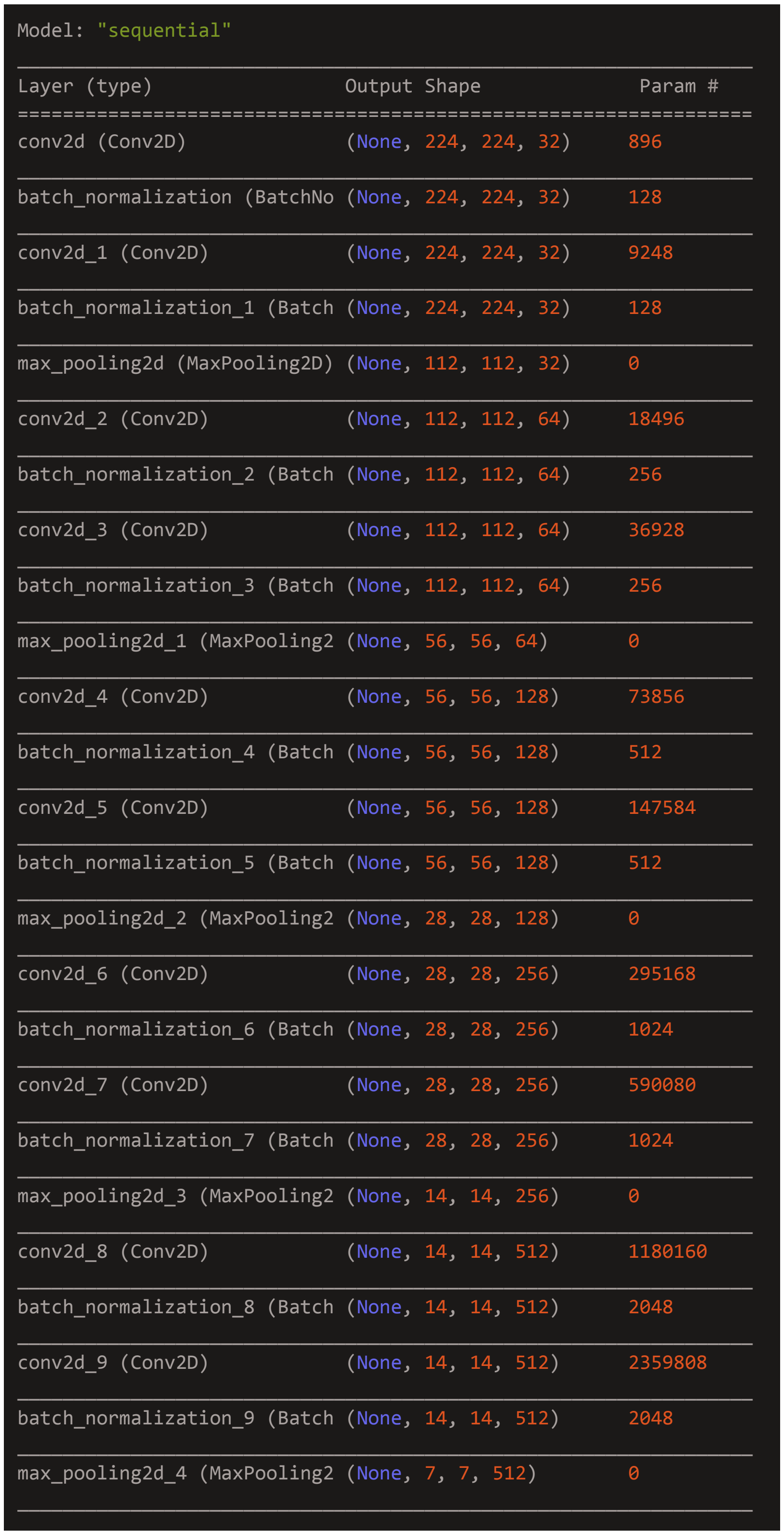
The sigmoid activation function is utilized in the output layer of the dense network for binary classification tasks. Sigmoid squashes the network's output into the range [0, 1], interpreting it as the probability of the input image belonging to the positive class (AI-generated image). This makes sigmoid particularly suitable for binary classification problems, where the goal is to assign a binary label to each input instance. By providing a clear probabilistic interpretation of the output, sigmoid facilitates decision-making and uncertainty estimation in the classification process. Additionally, sigmoid activation ensures that the model outputs are probabilistically meaningful, aiding in the interpretation and evaluation of the model's performance.

Dropout Regularization

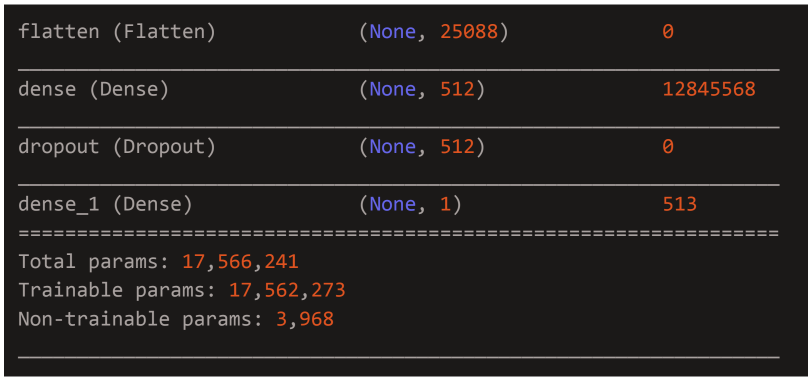
To prevent overfitting and improve generalization, dropout regularization is applied before the output layer. Dropout randomly deactivates a fraction of neurons during training, forcing the network to learn redundant representations and reducing the reliance on specific features.

3.2.2 Model Architecture

BINARY IMAGE CLASSIFICATION CNN SUMMARY



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 32)      896
_________________________________________________________________
batch_normalization (BatchNo (None, 224, 224, 32)      128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 224, 224, 32)      9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 224, 224, 32)      128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 64)      18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 112, 112, 64)      256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 112, 112, 64)      36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 112, 112, 64)      256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 56, 56, 128)       73856
_________________________________________________________________
batch_normalization_4 (Batch (None, 56, 56, 128)       512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 56, 56, 128)       147584
_________________________________________________________________
batch_normalization_5 (Batch (None, 56, 56, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 256)       295168
_________________________________________________________________
batch_normalization_6 (Batch (None, 28, 28, 256)       1024
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 256)       590080
_________________________________________________________________
batch_normalization_7 (Batch (None, 28, 28, 256)       1024
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 256)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 512)       1180160
_________________________________________________________________
batch_normalization_8 (Batch (None, 14, 14, 512)       2048
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 512)       2359808
_________________________________________________________________
batch_normalization_9 (Batch (None, 14, 14, 512)       2048
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 512)         0
_________________________________________________________________
```



```
flatten (Flatten)            (None, 25088)            0

dense (Dense)                (None, 512)              12845568

dropout (Dropout)            (None, 512)              0

dense_1 (Dense)              (None, 1)                513
=================================================================
Total params: 17,566,241
Trainable params: 17,562,273
Non-trainable params: 3,968
_________________________________________________________________
```

The CNN architecture consists of alternating convolutional and pooling layers followed by densely connected layers, producing a binary classification output with a probability of the image input being AI-generated. This architecture is designed to efficiently capture hierarchical features from the input images while minimizing overfitting and maximizing classification performance. The utilization of batch normalization and dropout regularization further enhances the model's robustness and generalization capabilities. It has 17,566,241 total parameters and 17,562,273 trainable parameters.

The resulting model strikes a balance between complexity and simplicity, enabling it to be effective in classifying AI-generated images, but also abide by our constraints around computational power (NVIDIA GeForce GTX 1650 GPU with 4GB of memory).

4. Results

4.1 Hyperparameters

The face classifier model was trained with the following hyperparameters:

- Batch Size: 32
- Image Size: 224x224 pixels
- Epochs: 10
- Optimizer: Adam

Batch size was set to 32 in order to strike a balance between stability and memory constraints. Image size was limited to 224 x 224 pixels due to the limited quality data that we could find, consistency with our model, and limited computing resources we have. The epochs were set to 10 given the convergence of both training and validation losses. Finally the optimizer was set to Adam as it is a default choice and is known for its adaptive learning rates and momentum properties.

The image classifier model followed a similar pattern. Here are the hyperparameters for the image classifier model:

- Batch Size: 16
- Image Size: 224x224 pixels
- Epochs: 10
- Optimizer: Adam

After trial and error we came to a conclusion to use a batch size of 16 for this model. This number allowed for a good balance between stability and memory constraints. The image size was limited to 224 x 224 pixels as well due to the limited amount of data, consistency of our models, and computing power issues. Epochs were set to 10 for both training and validation. Finally the optimizer was set to Adam.

4.2 Main Results

The results of the face classifier model were determined using training accuracy and validation accuracy:

- Final Training Accuracy: 98.82%
- Final Validation Accuracy: 98.08%
- Training Loss: 0.0349
- Validation Loss: 0.0614

Training History:
Here are the training loss, accuracy, validation loss, and validation accuracy in a table format for the face classifier model through the 10 epochs.

Loss	Accuracy	Validation Loss	Validation Accuracy
0.895644844	0.609979987	0.566042364	0.701900005
0.552488446	0.722469985	0.499188811	0.780799985
0.412680358	0.819429994	0.411865324	0.812300026
0.25865823	0.903699994	0.356503189	0.866450012
0.155344114	0.946399987	0.396919668	0.834800005
0.094690487	0.967140019	0.1005973	0.957099974
0.069315463	0.977039993	0.076931074	0.973349988
0.051306203	0.98259002	0.092519313	0.973950028
0.041600782	0.985289991	0.073294029	0.980250001
0.03494842	0.988179982	0.061438367	0.980799973

The training accuracy steadily increases with each epoch, reaching 98.82% by the final epoch. Both training and validation losses decrease significantly throughout the training process, indicating effective learning and model convergence.We can see that the model is able to perform extremely well without having major underfit or overfit issues.

The results of the image classifier model were determined using training accuracy and validation accuracy:

- Final Training Accuracy: 95.69%
- Final Validation Accuracy: 93.56%
- Training Loss: 0.0896
- Validation Loss: 0.1894

Here are the training loss, accuracy, validation loss, and validation accuracy in a table format for the image classifier model through the 10 epochs.

Loss	Accuracy	Validation Loss	Validation Accuracy
0.751404226	0.657130599	0.414012462	0.803411186
0.328613222	0.856442571	0.234398171	0.899670362
0.225830555	0.900601983	0.195697546	0.913802505
0.180971742	0.919263303	0.181041405	0.919678926
0.153293461	0.929554224	0.157337084	0.928278625
0.136970967	0.935459375	0.156433761	0.930199206
0.121399291	0.941973627	0.164964303	0.930170536
0.108042113	0.947549105	0.159408733	0.926788032
0.098821238	0.953303695	0.158378646	0.934441745
0.089601584	0.956908405	0.189360917	0.93561703

The training accuracy steadily increases with each epoch, reaching 95.69% by the final epoch. Both training and validation losses decrease significantly throughout the training process, indicating effective learning and model convergence.

4.2 Further Analysis

As we look at the two results generated from the models, we can see that both models achieved high accuracy on both the validation and test datasets, showing that there is good generalization. In addition, there are no issues with overfit models as the test accuracy and the validation accuracy are close to each other. The test and validation loss is low for both models as well, showing that there is a stable performance on new data. One issue that can be addressed is the size of the image 224 x 224. These image sizes are small and may not capture the full depth of an image properly in real life image examples.

5. Conclusion

With the increase of generative AI, images became increasingly difficult to detect whether they are AI generated or not. Our premise of the project was to help distinguish the difference between real and fake images. In this project we collected 2 labeled datasets: general images and another focusing on human faces. The images were resized to 224x224 pixels for consistency. Then 2 Convolutional Neural Network (CNN) architectures were designed. The first model was used to classify general AI-generated images and the second model was used to distinguish AI-generated faces. Each CNN consisted of five convolutional blocks followed by densely connected layers. These models were trained using supervised learning on the labeled datasets with specific hyperparameters such as batch size, image size, and epochs. These models were then evaluated using validation data, and their performance was assessed based on accuracy and loss metrics. Both models achieved high accuracy and low loss meaning that the models were both accurate and generalized. These models were extracted and tested using our streamlit and flask website, where you can input any image to determine whether the image is AI generated.

While our models were able to perform well and produce results in live testing, there are some areas for improvement. The image size used for the model training had to be 224 x 224 pixels which is considerably very small due to the lack of quality datasets and computing power. In order to tackle this issue, we would need to find and generate more quality data with higher resolution to capture finer details of images. In addition, having access to high computing power machines will help with faster training times for our modes. In addition, our website and models are not available to the public as we run the application in a local host. Having the website and model deployed to the public for access will bring in feedback and discussion to our models and help improve the project. Another aspect that needs more work is to explore different ensemble techniques to combine predictions from multiple models for enhanced accuracy and robustness.

References

[1]”Image Classification: A Survey,” *J. Infor. Electr. Electron. Eng., vol. 1, no. 2* https://jieee.a2zjournals.com/index.php/ieee/article/view/2 (accessed Apr. 13, 2024).

[2]”Review of deep convolution neural network in image classification,” *2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)* https://ieeexplore.ieee.org/abstract/document/8253139?casa_token=Zn-ULtJJfOcAAAAA:HQGvbp4uoKymqiaKApHT_l0-sQHBRnM9Ih-MWiNSV7qgdtAjJS_H6YaSU1hioTwLNt1BBFuBKw (accessed Apr. 13, 2024).

[3]“Realifake,” *www.kaggle.com*. https://www.kaggle.com/datasets/sattyam96/realifake (accessed Apr. 13, 2024).

[4]“140k Real and Fake Faces,” *www.kaggle.com*. https://www.kaggle.com/datasets/xhlulu/140k-real-and-fake-faces/data (accessed Apr. 14, 2024).

GitHub Link

https://github.com/Antonio-Villarreal/ai_classification_project