AI vs Human Generated Images

Blake Baker, Ryan Bowering, William Chong, Derick Miller, and Raymond Zhao

CSSE/MA415 Machine Learning

19 May 2023

ABSTRACT

This paper presents a study on identifying AI-generated images from real ones using a variety of classification models and feature engineering including VGG16. The study used the CIFAKE dataset and achieved over 90% accuracy in classifying the images as either real or AI-generated, which is significant considering the increasing difficulty of distinguishing AI-generated images from real ones. However, the study is limited by the use of only one CNN architecture, VGG16, for feature extraction, and the lack of feature explainability. The paper suggests using other CNN architectures and incorporating Attention into the model to increase accuracy and explainability in future studies. Additionally, the paper discusses the challenges faced during the study, including dataset formatting, long training times, and resource limitations, and offers suggestions for optimization in future studies.

1.INTRODUCTION

With the introduction of artificially created images, it has become increasingly difficult to discern between human and artificially generated pictures. The problem of differentiating between AI and human generated images is an interesting problem and has a wide range of applications. By accurately distinguishing between the two, we can combat the spread of misinformation and deepfakes to improve the reliability in information we consume. This problem is especially relevant in the context of moderation on online platforms, where it can help ensure the authenticity and quality of user-generated content, leading to safer online environments. The development of a dependable algorithm relies heavily on concepts from machine learning. We implemented various classification techniques such as logistic regression and support vector machines to analyze image features and train an algorithm to distinguish between human and computer generated images.

However, the problem of differentiating between these images presents several challenges that make it a significant time investment. A significant challenge we encountered was the sheer size of the data set. Working with even a small portion of the data resulted in long processing times which slowed down progress, even reaching a predicted run time of over 24 hours at times. Although the concepts attributed to this project stem from machine learning, there was still additional knowledge the team needed to learn before work could be done. Examples include learning how to decompose image files into arrays and understanding the processing behind the VGG16 method.

Significant outcomes were obtained from this project as running support vector machines on the VGG16 extracted features resulted in a testing accuracy rate of over 90%. This was the highest validation score we achieved, greatly outperforming other approaches we took and the baseline accuracy of 50%. This result is also comparable to the data set's original author who utilized a small Convolutional Neural Network to process and predict on the data which achieved an accuracy of 92%.

2. LITERATURE REVIEW

While the popularity spike of AI image generation is fairly recent, there has been some work done to train models to distinguish between real and AI generated images. Bird and Lofti used a stable diffusion model to generate counterparts to the CIFAR-10 dataset of images and called the combined dataset CIFAKE [1]. This is the dataset we used for this paper. Additionally, they trained a Convolutional Neural Network to classify the images, and used Gradient Class Activation Mapping to view which parts of an image had the largest impact on their model's prediction. They were able to achieve 92% accuracy and found that real images had a fairly uniform distribution of importance across each image while

the important sections of AI generated images were much more separated [1]. During our research, Piosenka trained a model using transfer learning and an EfficientNetB0 model on the CIFAKE dataset and achieved an accuracy of 96% [2].

An additional subject of note in previous works was that of SRM filtering. In particular, one paper by X. Chang et al. used an SRM kernel to extract noise data from deepfake images to classify them as real or fake [3]. The authors used VGG16 as an initial test to classify the deepfakes but due to poor results implemented an SRM filter on top of it as a custom VGG16 kernel. Our work follows a similar process to the paper and was influenced to attempt a similar experiment to see if the same technique could be utilized on purely real and fake images. The process was also different, as we started with running images through an SRM filter instead of using VGG16 first. The paper also influenced our use of VGG16 for the image overall.

The previous paper, in addition to a paper by M. Hussain, J. Bird, and D. Faria on CNN transfer learning [4] introduced the use of transfer learning to our overall process. While the previous paper introduced VGG16, the idea to use the tool for transfer learning came from this second paper. While we decided not to use a CNN at any point in our work, the paper's process could be used with little modification to our process and could potentially improve our results. As a result, the team used VGG16 in a similar way and attempted the use of transfer learning to classify real and fake images.

3.PROCESS

3.1 Data Source

The data is made up of 32x32 RGB images that were publicly available on Kaggle [1] [6]. The dataset contains a total of 120,000 images, with 60,000 real images and 60,000 fake images.

The real images are taken directly from the CIFAR-10 dataset and consist of 60,000 color images in 10 classes, with 6,000 images in each of the following categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The fake images were generated using Stable Diffusion version 1.4 to and has the same distribution that the CIFAR-10 dataset does.

These datasets are split equally so that there are 100,000 training images (50,000 per class) and 20,000 testing images (10,000 per class). We are then using this data set to train our own classification model to predict whether images are of two classes, real or AI modified/generated. As the data was already split evenly, the two classes are balanced and have the same number of targets so there should not be any issues in this regard.

3.2 Preprocessing

The images were downloaded as jpg files which needed to be converted into numerical values to perform the necessary learning algorithms. After loading the data, we used the Python Imaging Library to extract the data into 32x32x3 arrays which represent the 32x32 pixel image with RGB three color channels.

To set up the baseline logistic regression, we initially tried to put the 32x32x3 arrays into the logistic regression function from the Scikit Learn package in python. This ended up printing error messages since the function did not have tensor support. As a result, the images were flattened into a 1x3072 array to support the logistic regression. This classifier treats each pixel and RGB layer in every image as its own feature to learn the relationships between pixel values and make predictions based on these relationships. Finally, the pixels were standardized before running through the logistic regression. Contrary to the belief that standardization would not affect the results since we are looking at each pixel individually, we found that we had faster training times and higher R2 scores when we standardized the data.

3.3 Feature extraction

One of the main aspects of our project was seeing how different models may improve the classification and so we implemented two methods of feature extraction prior to any fitting. The first was VGG16 which is a convolutional neural network (CNN) architecture that was developed by the Visual Geometry Group at the University of Oxford. It is widely used for image classification tasks, and has achieved state-of-the-art performance on a number of benchmark datasets.

At a high level, the VGG16 network works by taking an input image and passing it through a series of convolutional and pooling layers to extract features from the image. The features are then fed through several fully connected layers to make a prediction about the class of the image. Since we only wanted to use the feature extraction aspect of VGG16, we removed the fully connected layers that would make the predictions and implemented our own methods including random forests and gradient boosted trees.

The second form of feature extraction was the steganalysis rich model (SRM). When an AI generates an image, there are often subtle differences between the AI-generated image and a real-world image. These differences can be detected by analyzing the features of the image using machine learning algorithms. These features can be based on the statistical properties of an image, such as the mean, variance, and higher-order moments of the pixel values. SRM uses statistical features among a variety of other relationships to detect deviations from the expected distribution of pixel values in a cover image, which can indicate the presence of hidden information and noise.

An SRM filter was used on the images as part of an attempt to see if noise data alone would be able to accurately predict whether an image is real or fake. Any accuracy over purely guessing would be a success for this set of features.

These feature extractions were not done because they were required, but to see how they would perform and whether a better result could be derived for them. Their utility and effectiveness will be further discussed in the next section along with the models they were implemented in.

3.4 Classifiers/regressors and tuning

This project is purely a classification oriented task, aimed at differentiating human and AI generated images. We approached this problem with various models: logistic regression, random forests, gradient boosted trees, and support vector machines.

Logistic regression is a classification algorithm that predicts the probability of an instance belonging to a certain class. This is accomplished by modeling the relationship between input features and the probability of class membership. Logistic regression assigns each class its own linear regressor and learns the weights of the linear equation during the training phase to minimize the difference between probabilities of the training data. The primary hyperparameters for logistic regression were introduced when regularization was added. We looked through L2 and L1 regularization as well as various regularization strengths. The Lasso and Ridge regularization added a penalty term to the loss function, introducing a trade-off between the model's ability to fit the training data and its complexity. This is used to reduce overfitting of the training data. Additionally, regularization strength will determine the strength of the penalty term. Tuning the hyperparameters did not see significant results as the highest scores were seen by the L1 regression with a validation score of 0.665 and a test score of 0.666.

Random forest is a model that uses a collection of decision trees to make predictions. Each tree is trained on a random bootstrap sample, and they are all very deep to reduce bias as much as possible. The predictions of all the trees are averaged to produce the final prediction to control the variance of the model. Random Forests have two hyper parameters: the number of estimators and the maximum depth. We chose to only optimize the number of estimators since adding extra dimensionality to our grid search increased training times. 5-fold cross validation was used to score each hyper parameter. The results of our grid search are summarized in Figure 1. The validation score of our model leveled off after 100-200 estimators so we are fairly confident that our hyperparameters are optimal. Random Forests achieved a validation score of 0.869 and a test score of 0.8715 on our data. The best hyperparameter was 440 estimators.

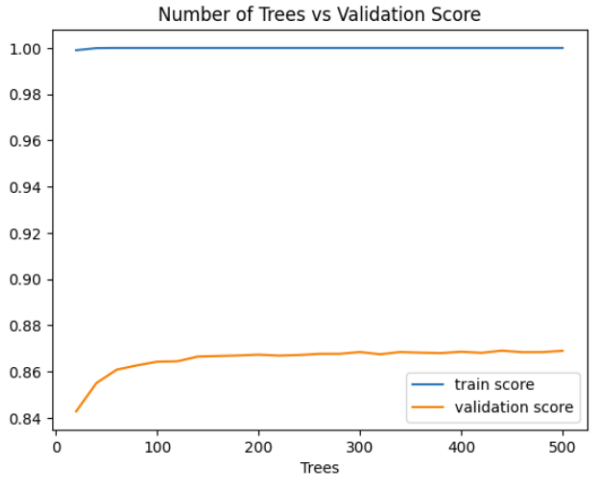

Figure 1: Plot of the validation score for different numbers of estimators for a Random Forest classifier.

Gradient Boosted Trees is another model that uses a collection of trees, but unlike Random Forests each tree has a very low depth to limit variance. To minimize bias, each sequential tree is trained on the residuals of the previous tree to minimize the difference between the predictions and the target over time. Gradient Boosted Trees have max depth and number of estimators as hyper parameters like Random Forests, and they also have a learning rate which controls how much each tree can affect the prediction of the model. We optimized all three in our model with 5-fold cross validation since we were able to accelerate it with GPUs. We used two separate grid searches to look at a wider range of estimators. Figures 2 and 3 summarize these results. Learning rates of 0.1 and 0.5 performed the best across the board, max depth seemed to perform similarly for values 3 and 4, and number of estimators started converging at around 2000. We achieved a validation score of 0.8994 and a test score of 9043 on our data. The best hyper parameters were 2900 estimators, learning rate of 0.5 and max depth of 4.

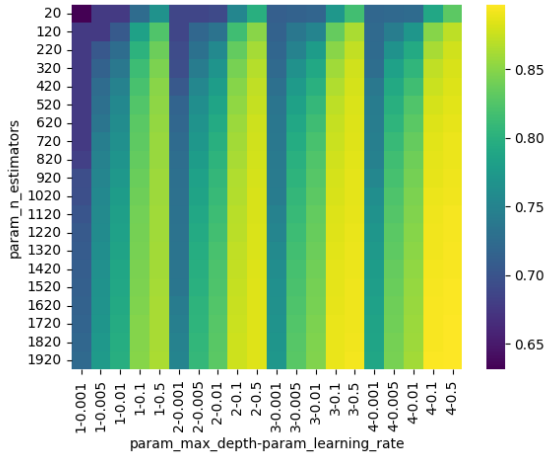

Figure 2: Heat map of validation scores for different combinations of max depth and learning rate (x axis) and number of estimators (y axis) for a Gradient Boosted Tree classifier. Lighter values correspond to higher scores.

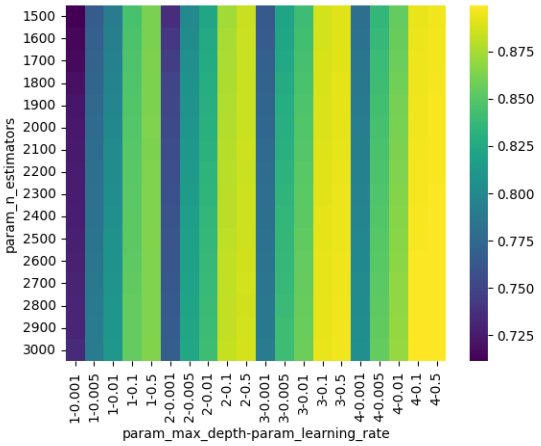

Figure 3: Heat map of validation scores for higher numbers of estimators for a Gradient Boosting Tree classifier.

Support Vector Machines is a model that attempts to find the optimal hyperplane separating the data into two classes. It does this by trying to maximize the margin between the hyperplane and the nearest data points which are called support vectors. The hyperparameters of this model include C which determines how large the margin can be, gamma which determines how smooth the decision boundary is, and the kernel which determines what type of function the decision boundary can be. We experimented with both a sigmoid and radial basis kernel, and the radial basis function gave us the best

results. We used 5-fold cross validation to optimize C and gamma. Figure 4 summarizes the results of this grid search. We found that a fine grid search gave a small range of possible hyperparameters that achieved similar validation accuracy, so we aired on the side of smaller values for C and gamma since larger values run the risk of overfitting. We achieved a validation score of 0.905 and a test score of 0.91.

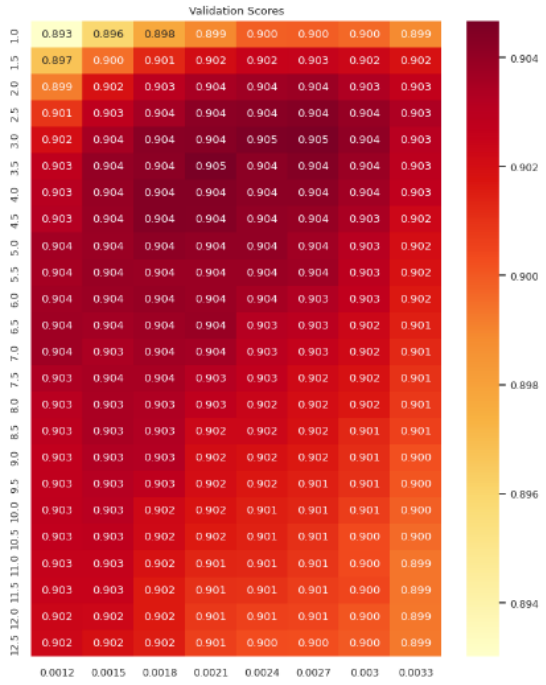

Figure 4: Heat map of validation scores for different combinations of C (x axis) and gamma (y axis) for a Support Vector Machine classifier. Darker values correspond to higher scores.

3.5 Post-processing

We thought it would be interesting to see which pixels are the most impactful on how our model determines whether an image is fake (AI) or real (human). We started with a logistic regressor using the direct pixels as it would be easy to pull the pixel importance from the model if each pixel was a feature. Taking the model's coefficient weights, we were able to create a heat map of which pixels were the most important in determining image authenticity.

Figure 5 shown below, displays one of these images for lasso regularization. It can be observed that the most important pixels were generally found to be in the corners/edges and in the center of the image.

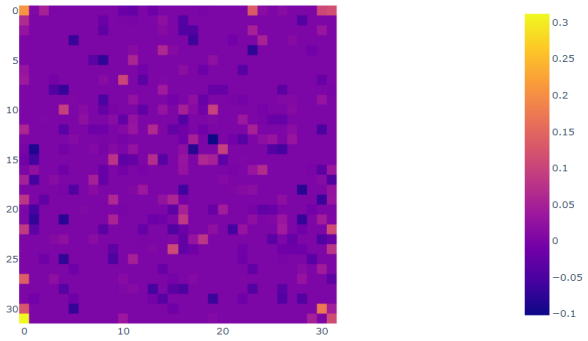

Figure 5: Heat map of pixel importance weights calculated by logistic regression with lasso regularization. Lighter values correspond to higher scores.

More post-processing should be done on the more complicated models using a variety of feature extraction. While it is easy to see how these specific pixels may have a correlation with whether an image is real or fake, more abstract correlations and features are harder to define and understand from a model. If more time was available we would have attempted to gain a better understanding of the features created through VGG16 and how they may be defined by humans rather than a collection of numbers.

4. EXPERIMENTAL SETUP AND RESULTS

The data set contains a total of 120,000 images with an even split of 60,000 real and 60,000 computer generated images collected from CIFAR-10. Shown in Table 1 is the distribution of the images in the training and testing sets. These sets also maintained the even split between real and fake images, which set the pure baseline to a 0.5 accuracy rate.

Table 1: Distribution of real and fake images in the training set and testing set downloaded from Kaggle [1].

Data Set	**Real**	**Fake**
Training	50,000	50,000
Testing	10,000	10,000

We decided to start with a logistic regression approach, where each pixel was treated as an individual feature, to establish a higher baseline

compared to random guessing. To conserve time and processing resources, we utilized a sample of 10% from the total training set, consisting of 10,000 images. Three types of logistic regression were used on the training data: logistic regression without tuning hyper-parameters, L1 regularization, and L2 regularization. The results of each model are summarized in Table 2.

Table 2: Results of the Logistic Regression

	No Tune	**L1 Reg**	**L2 Reg**
Train	0.7264	0.6955	0.6946
Validation	0.6312	0.6651	0.6635
Test	0.6515	0.6661	0.6658

Next, we added VGG16 feature extraction before fitting our logistic regressor. This utilized the full dataset and also performed the three types of logistic regression as described previously: no hyperparameter tuning, L1 regularization, and L2 regularization. The results of the models are summarized in Table 3.

Table 3: Results of VGG16 Logistic Regression

	VGG16	**L1 Reg**	**L2 Reg**
Train	0.8595	0.8597	0.8599
Validation	0.8566	0.8564	0.8567
Test	0.8544	0.8547	0.8547

We then trained a Random Forest classifier on our VGG16 feature extracted data since they are accurate and hard to overfit. Since Random Forests are supposed to use large trees, we decided not to limit the max depth of our trees and only optimized the number of estimators. We ran a grid search from 20, 40, 60,…, 500 estimators. Figure 1 summarizes the results of this grid search. We were able to achieve a validation accuracy of 0.869 with 440 estimators. This model achieved a testing accuracy of 0.8715 which is better than the baseline of 0.5 and the logistic regression which scored about 0.66.

We also trained a Gradient Boosted Tree classifier on the same data. We optimized the hyperparameters for max depth, number of estimators and learning rate. We tested values of 0.001, 0.005, 0.01, 0.1 and 0.5 for learning rate, 1, 2, 3, 4, 5 for max depth, and 20, 120, 220,..., 1920 for number of estimators. We wanted to see how much better it could do with more estimators so we then ran a second grid search with the same values of learning rate and max depth but with of 1500, 1600, 1700,..., 3000 estimators. We found that higher learning rate and more estimators seemed to have the largest impact on the results. Figures 2 and 3 summarize the results of these grid searches. We were able to achieve a validation accuracy of 0.8994 with a learning rate of 0.5, a max depth 4 and 2900 estimators. This model achieved a testing accuracy of 0.9043 which is better than the baseline of 0.5 and the logistic regression which scored about 0.66.

Lastly, we trained a Support Vector Machine classifier on our data. We tried both a sigmoid and a radial basis function for the kernel on low sample sizes and found that the radial basis function gave the best results. We optimized the C hyperparameter which affects the size of the margin and gamma which affects the smoothness of the decision boundary. We tested values 1, 1.5, 2,..., 12.5 for C and values of 0.0012, 0.0015, 0.0018,..., 0.003 for gamma. We found that the optimal combinations of C and gamma converged to a small area. Figure 4 summarizes the results of this grid search. We were able to achieve a validation accuracy of 0.905 with values of 3.5 for C and 0.0021 for gamma. We achieved a testing accuracy of 0.91 which is better than the baseline of 0.5 and the logistic regression which scored about 0.66.

After we scored all of our models on the test data, we established what human performance would score classifying images as real or AI. To do this, we randomly selected 16 images, consisting of real and AI images and asked humans to classify them. We found that humans score an average accuracy of 0.525, only slightly better than the 0.5 baseline.

The SRM filter used on the set of images created a new set of filtered images that would show the noise levels of each pixel in the original image. Running

the images through the previously mentioned logistic regressor resulted in the classifier randomly guessing the class of the image. This gave an accuracy of 50%, which is the same as just guessing all of the images are one class. After these results, steps were taken to feed the new filtered images through VGG16 as well. Using the new features from VGG 16, the classifier was rerun. The results of the two runs are summarized in table 4.

Table 4: Results of SRM Filtering

Model:	SRM	SRM + VGG16
Train Accuracy:	0.500	0.6181
Valid Accuracy:	0.500	0.6111
Test Accuracy:	0.500	0.5733

5. DISCUSSION

The results of the baseline logistic regression present a higher accuracy than the baseline accuracy of 0.5 and human baseline accuracy of 0.525. Training each pixel as an individual feature in logistic regression helps simplify the representation of the image data, allowing the model's input to be better understood compared to the VGG16 extracted features. Additionally, this approach allows for fine-grained analysis, enabling the capture of subtle color, texture, and spatial variations within the image. Furthermore, by considering pixels individually, the model has the potential to capture localized patterns and variations, which can be advantageous for tasks where the arrangement or location of features is significant.

However, there are also drawbacks to treating each pixel as a feature. One major drawback is the resulting high dimensionality, which can lead to significant computational complexity, overfitting risk, and the curse of dimensionality. Moreover, the lack of contextual information and dependencies between neighboring pixels may limit the model's ability to understand overall image composition.

The results of our Random Forest, Gradient Boosted Trees and Support Vector Machines all scored significantly better than both the baseline and logistic regression. The downside of all these models is that we trained them on the VGG16 extracted features, so their results were much more abstract and not as explainable as the logistic regression. We generated confusion matrices for each of these regressors, summarized in Tables 5, 6 and 7.

Random Forests had more false negatives than false positives, implying that it was harsher on real images, predicting more of them to be fake. This might actually be a favorable bias for a predictive model since images can be used to frame people for things they didn't do so it could be beneficial for a model to err on the side of caution.

On the other hand, Gradient Boosted Trees had more false positives, than false negatives, meaning it was lenient on real images, classifying more fake images as real. This type of bias is likely not ideal in a predictive model as it will allow more fake images to pass for real which has worse consequences than falsely classifying an image as fake.

SVM had the same number of false positives as false negatives which implies that it performs similarly for all types of images. This type of model would be useful to make fair, unbiased predictions.

Table 5: Confusion matrix for Random Forests using VGG16

RF	Target		
Prediction		Real	Fake
	Real	8636	1207
	Fake	1364	8793
Precision: 0.8774 **Recall:** 0.8636			

Table 6: Confusion matrix for GBT using VGG16

GBT	Target		
Prediction		Real	Fake
	Real	9020	1022
	Fake	980	8978
Precision: 0.8982 **Recall:** 0.9020			

Table 7: Confusion matrix for Support Vector Machines using VGG16

SVM	Target		
Prediction		Real	Fake
	Real	9104	896
	Fake	896	9104
Precision: 0.9104 **Recall:** 0.9104			

The results of the SRM Filtering gave an interesting insight in regards to how much data is needed for models to accurately predict what images are real or fake. While the basic attempt with the pure image failed to produce any noteworthy results, the noise data run through VGG16 allowed for a higher accuracy in the logistic regressor. While the results were minimal, the fact that the noise data gave the classifier enough information to influence the accuracy of the prediction is a success.

Most of our machine learning models outperformed human performance, which scored a 0.525. The only model that did not outperform the human baseline was the logistic regression training only on the SRM filtered data. However, all of the models trained on the RGB data outperformed humans suggesting the machine learning models are more effective than humans at classifying real and AI images, particularly when trained on RGB data.

6. CONCLUSIONS AND FUTURE WORK

In this paper we found that images in the CIFAKE dataset could be classified as real or AI generated with over 90% accuracy. This confirms the findings of Bird and Lofti who trained a Convolutional Neural Network (CNN) that achieved 92% accuracy [1]. This is significant as AI generated images are becoming increasingly more difficult for humans to distinguish from real images, so a reliable method to identify AI generated images is necessary.

We limited our work to using the pretrained VGG16 CNN architecture [6] to extract features from the data and trained our own external models on those features. Piosenka used an EfficientNet model and achieved 96% accuracy which suggests that different CNN architectures could make more accurate classifications. Evolutionary algorithms such as Natural Evolution of Augmented Topologies (NEAT) could be used to optimize these architectures.

Furthermore, our work did not look into feature explainability. This is an important part of identifying AI generated images because understanding the features that a model uses to classify images might be able to help humans distinguish them on the fly. Using Gradient Class Activation Mapping, Bird and Lofti found that when classifying an image as real, most of the image was important, but the important defining features of AI generated images were more sparse.

For future work, incorporating Attention into a model could help increase both the accuracy and explainability of predictions, but no work has been done with this for identifying AI generated images.

Finally, the results of the SRM filtering indicates that the creation of features from images using filters is viable. The features created can then be used in classifiers to boost their accuracy. While the noise filter did little to boost the accuracy, other filters likely can reveal more about the image and when used in conjunction with the original pixel data, can likely improve the accuracy of classification.

7. KEY CHALLENGES AND LESSONS LEARNED

We faced several challenges during this project. First, our dataset was a massive collection of images which meant that it was formatted as tensors, which our Scikit Learn models are not equipped to handle. Thus, it took us a while to format the data to be usable. Furthermore, since raw pixel values aren't very useful, we needed to use VGG16 to extract features, which meant that our results were not very explainable.

Second, Our dataset was very large so we had very high training times on the full dataset. We needed to learn how to accelerate our code with GPUs to even finish training some of the more computationally intensive models like Gradient Boosted Trees and Support Vector Machines. It was also extremely difficult to predict training times for the full dataset from smaller samples. This meant that it would often take much longer to train models than we expected which caused several timeouts without any results.

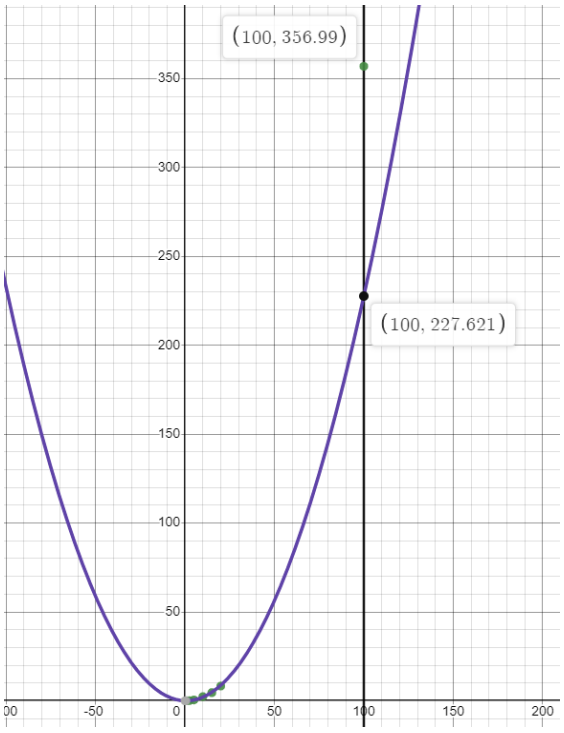

Figure 6: A graph predicting the training times of an SVM grid search from samples of 1, 2.5, 5, 10, 15 and 20% of the full dataset, and the time it took to train the full dataset (356.99 minutes).

Lastly, we ran into many issues finding the resources to train our more computationally expensive models. We received a warning from the server administrator about using CPU resources on the GPU server, and we had that job shut down after running it on the CPU server for too long. We were unable to install the necessary packages on the GPU server so we were forced to switch to Kaggle's servers where we were limited to 30 hours per week using the GPUs and 12 hour sessions.

We learned that in the future we need to be more concerned with optimization and start with smaller grid searches to narrow in on hyperparameters before running finer grid searches so we don't waste as much computation time. We also learned that we need to plan for training to take longer than we expect so that we don't run into situations where we use too many resources on school servers or time out on external servers.

8. REFERENCES

[1] Bird, J.J., Lotfi, A. (2023). CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images. arXiv preprint arXiv:2303.14126. Available: https://arxiv.org/pdf/2303.14126.pdf

[2] Piosenka, J. (2023). Transfer Learning F1 score=96% [Jupyter Notebook]. Available: https://www.kaggle.com/code/gpiosenka/transfer-learning-f1-score-96

[3] X. Chang, J. Wu, T. Yang and G. Feng, "DeepFake Face Image Detection based on Improved VGG Convolutional Neural Network," *2020 39th Chinese Control Conference (CCC)*, Shenyang, China, 2020, pp. 7252-7256, doi: 10.23919/CCC50068.2020.9189596.

[4] M. Hussain, J. Bird, D. Faria. "A Study on CNN Transfer Learning for Image Classification". *Aston University*. June, 2018.

[5] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images

[6] Simonyan, K, and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.15 Available: https://arxiv.org/abs/1409.1556

[7] OpenAI. "ChatGPT." OpenAI, 2021. [Online]. Available: https://openai.com. [Accessed: May 14, 2023].

[8] Zhou, P., Han, X., Morariu, V. I., & Davis, L. S. (2018) .Learning Rich Features for Image Manipulation Detection. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. doi:10.1109/cvpr.2018.00116

9. APPENDIX

Appendix A: Individual Example (Predicting on GBT)

In this section, we provide an overview of the code to use a trained XGBoost classifier on the CIFAKE dataset, and predict whether a sample image is AI or human created. The code was written on kaggle in order to take advantage of the GPU resources offered by kaggle.

The following software packages were used for training the XGBoost classifier:

- pandas : for data manipulation and analysis
- numpy : for numerical computing
- xgboost: for implementing the XGBoost classifier
- tensorflow: for GPU acceleration, and VGG16
- scikit-learn: for preprocessing

```
import pandas as pd
import numpy as np
import xgboost as xgb
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input,VGG16
from sklearn.preprocessing import LabelEncoder
from time import time
from PIL import Image
```

Figure 7: Necessary software package imports

After importing our software packages we will load in our VGG16 feature extracted training set, and train the GBT classifier. We have already optimized the GBT classifier hyperparameters so we already know we will fit the classifier to the training set with the optimized hyperparameters.

```
df_real = pd.read_csv("/kaggle/input/cifake-vgg16-extracted-features/REAL_Train_Features.csv")
df_fake = pd.read_csv("/kaggle/input/cifake-vgg16-extracted-features/FAKE_Train_Features.csv")
df = pd.concat([df_real,df_fake],axis=0).drop('Unnamed: 0', axis=1)

df_sample = df.sample(frac=1.0, random_state=0)

X = df_sample.drop('class', axis=1)

le = LabelEncoder()
y = df_sample['class']
y = pd.DataFrame(le.fit_transform(y), columns=['class'])  # encode y to be numeric
```

Figure 8: Reading in VGG16 extracted features from CIFAKE training set, which were stored in .csv to a dataframe

```
gbt = xgb.XGBClassifier(learning_rate= 0.5, max_depth= 4, n_estimators= 1920,tree_method='gpu_hist',
    return_train_score=True
)
```

Figure 9: Creating a GBT classifier instance with hyper parameters: learning rate = 0.5, max_depth = 4, n estimators = 1920

After loading in the dataset and creating an instance of the GBT classifier with optimized hyperparameters we can now fit the classifier to the training data. To do this we use tensorflow to utilize the GPU on Kaggle's server to accelerate our training time.

```
with tf.device('/GPU:0'):
    time_start = time()
    gbt.fit(X, y)
    time_stop = time()

    print(f"Time elapsed: {(time_stop - time_start) / 60.0:.2f} minutes")
```

Figure 10: Using tensorflow to access the GPU on Kaggle's server to fit our GBT classifier on

Now that we have a fully trained GBT classifier, we load our sample image in resized as a 32,32 image to match our training set. We then standardize the image pixel data and pass it through VGG16 to extract features from the RGB pixel data.

```
im = image.load_img("/kaggle/input/demo-image-ship/ship.jpg",target_size=(32,32,3))
im = image.img_to_array(im)
```

Figure 11: Loading in the test image as 32x32 px RGB image and turning it into an array so we can perform feature extraction

```
img_height = 32 # The dataset is all 32px but this is here just to make sure
img_width = 32
feat_out = np.zeros((1,32,32,3))
model = VGG16(weights='imagenet', include_top=False)
# get the path/directory
count = 0
folder_dir = '/kaggle/input/demo-image-ship/'
for images in os.listdir(folder_dir):
    img_path = folder_dir+images
    img = image.load_img(img_path, target_size=(img_height, img_width))
    x = image.img_to_array(img)
    x = np.expand_dims(x,axis=0)
    mean = np.mean(x, axis=(1,2), keepdims=True)
    std = np.std(x, axis=(1,2), keepdims=True)
    standardized_images_out = (x - mean) / std
    feat_out[count] = x[0]
    #print(count)
    count = count + 1
print("starting preprocess")
feat_out = preprocess_input(feat_out)
features = model.predict(feat_out)
print(features.shape)
    #feat_out.append(features[0][0][0])
#feat_out.shape

feat_out.shape

store = np.zeros((1,512))
for i in range(1):
    store[i] = features[i][0][0]
demo = pd.DataFrame(store)
demo['class'] = 'Fake'

demo.head(3)
```

Figure 12: Standardizing the image array, extracting features with VGG16, and then sending values to a dataframe to be stored

Lastly all we need to do is separate the features from the target or class, in our case. Then, predict the target, or class, on the features.

```
X_demo = demo.drop('class',axis=1)
y_demo = demo['class']
```

Figure 13: Separating the extracted features from the target

```
le.inverse_transform(gbt.predict(X_demo))
```

```
array(['Fake'], dtype=object)
```

Figure 14: Passing the extracted features into the GBT classifier to receive the prediction