Classifying AI Generated vs. Real Images

Alejandro Rea

1 Problem Statement

The recent advances of image generation/synthetic data raise a serious concern about data validty and being able to discern between real and fake images is a crucial task. To do so, this report covers image calssification of AI generated images (fake) and real images using traditional ML methodologies and then a deep learning approach using a deep CNN. The data portion will dive into the justification for a CNN.

2 Data

We classify fake vs. real images using the CIFAKE dataset which contains 50000 fake images and 50000 real images in the train set. The test set contains 10000 fake and real images. The images are 32x32 pixels and of RGB values. Although the dataset is a clean even split of fake and real images, transferring the data lost some images due to a limit of images that could be transferred to the IDE. The real class distribution for the training data is found below (0 for fake and 1 for real).

The first attempt at image classification was through a random forest that decides wether an image is AI generated (fake) or a real image. To train a random forest on spatial data (images), statistics about the images were extracted and formatted into a dataframe. The statistics were not too highly correlated with each other with the exception of mean and median intensity nor did the distribution of each statistic have any erratic distributions (all skewed or normal).

The random forest, however, was not the appropriate tool for image classification as it reported a 1.00 accuracy on the training data and a 0.60 accuracy on the test data. Therefore, the model is extremely overfit and will not handle unseen data well. This is likely due to the fact that the random forest is not able to capture the spatial relationships between pixels in an image. For this reason, a CNN was used to classify the images.

All Data EDA and RF performance is shown below.

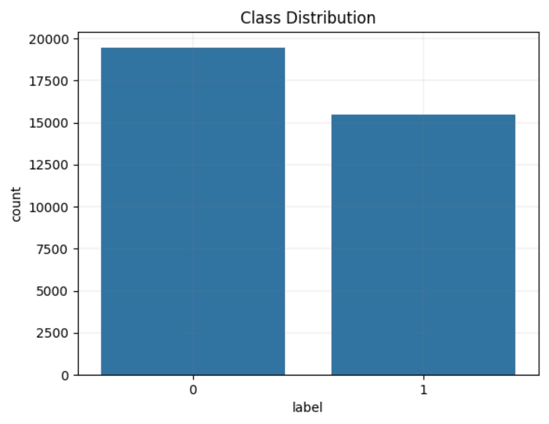

Figure 1: Train Class Distribution

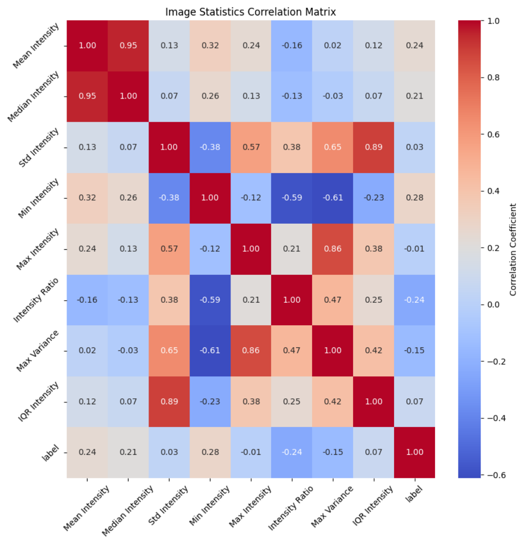

Figure 2: Correlation Matrix of Image Stats

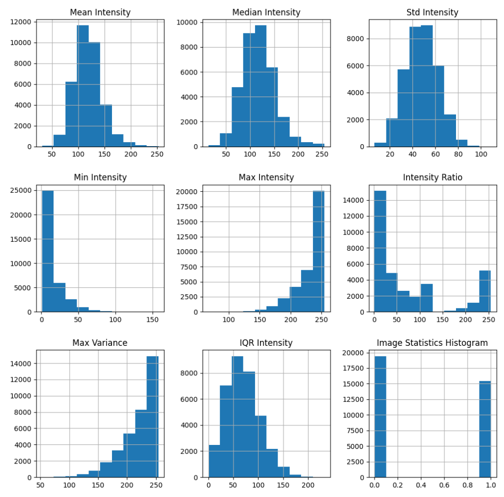

Figure 3: Statistics Distributions

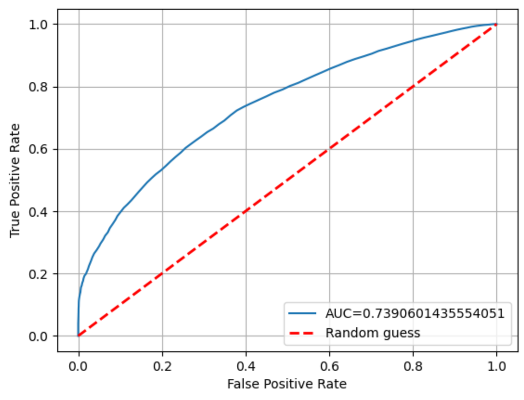

Figure 4: RF ROC Plot

3 Methods (Processing)

For the CNN, the data came pre-split in train and test folders, with train accounting for 80% of the data. The images were loaded into the model using the ImageDataGenerator class from keras, normalized and slightly augmented (shear range of 0.2). All images were kept at 32x32 with 3 channels. To actually load in the images as tensors, the flow_from_directory method was used.

4 Methods (Model)

The CNN is a deep network that closely mirrors VGG16's architecture. That is, it has 4 "blocks" that consist of convolutional layers, max pooling, batch normalization, and dropout in that order. Specifically speaking about only the conv layers of each block, the model starts with a 32x32x3 input shape and the first block has two conv layers (64 filters), the second block has two conv layers (128 filters), the third block has three conv layers (256 filters), and the fourth block has three conv layers (512 filters). The convolutional layers have a kernel size of 3x3, a stride of 1, same padding, and ReLU activation while the max pooling layers have a pool size of 2x2. The output is then flattened and passed through three dense layers (of size 4000, 1500, and 1 respectively). The dense layers also employ ReLU activation with the exception of the output layer using the sigmoid activation function. To compile, The model uses the Adam optimizer, binary crossentropy loss, and accuracy, precision, and recall as the metrics. The model was trained for 350 epochs with a batch size of 32. Below is a rough reference of what the model looks like found from this website discussing VGG16.

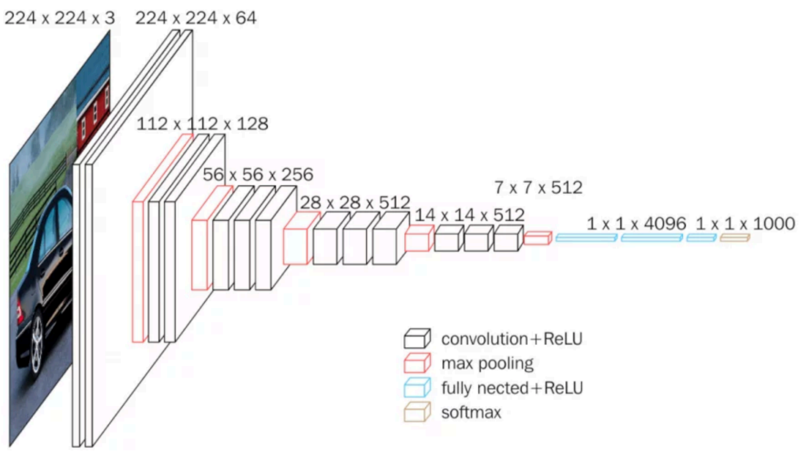

Figure 5: Model Architecture Reference

5 Results

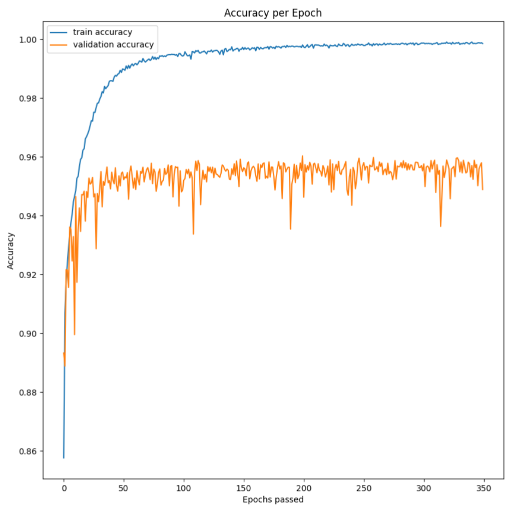

Figure 6: Accuracy per Epoch

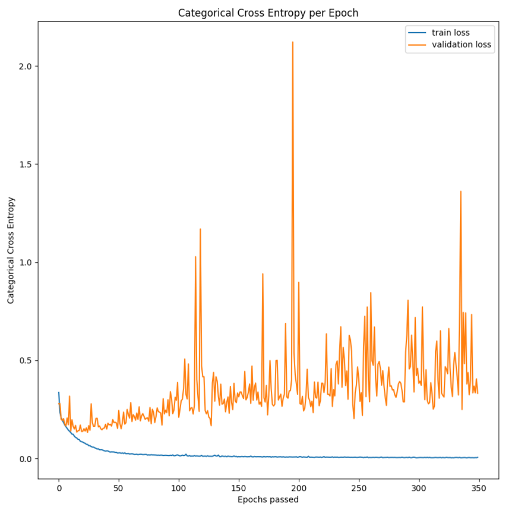

Figure 7: Loss per Epoch

The model was trained for 350 epochs and the accuracy and loss per epoch are shown above. The model achieved a 0.99 accuracy on the training data and roughly 0.95 accuracy on the test data. This already far outclasses the performance of the random forest as we are still reaching an extraordinarily high accuracy on the training data but the model is not overfitting. This is due to the fact that the CNN is able to capture the spatial relationships between pixels in an image through the kernels used to extract features. The hierarchical feature extraction allows the model to classify fake vs. real on unseen images. The model also has a high precision and recall on the test data, with a precision of 0.98 and a recall of 0.92. This means that a prediction of true has a 98% of being right.

6 Plots/Figures

Below are the plots and figures that further dive into the performance and explainability of the CNN. This provides insight as to what the model is predicting. For example, the images captioned by what the model classifies it as vs. its real label shows how the model is able to predict the truth for both classes with high accuracy. The confusion matrix showing that the model is more likely to incorrectly classify a fake image as real. A possible explanation for the more frequent prediction of class 0 (even if it is truly class 1) could be due to the previously mentioned slightly higher amount of class 0 images.

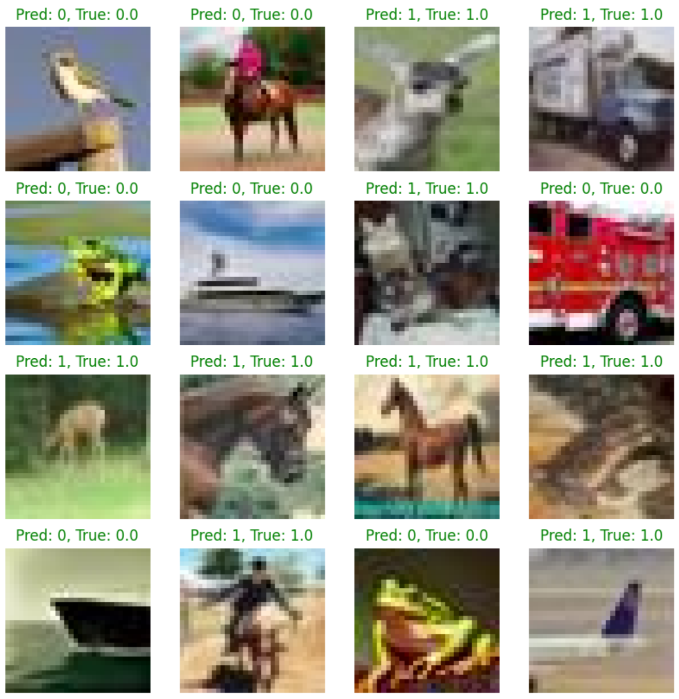

Figure 8: Predictions vs. True Label

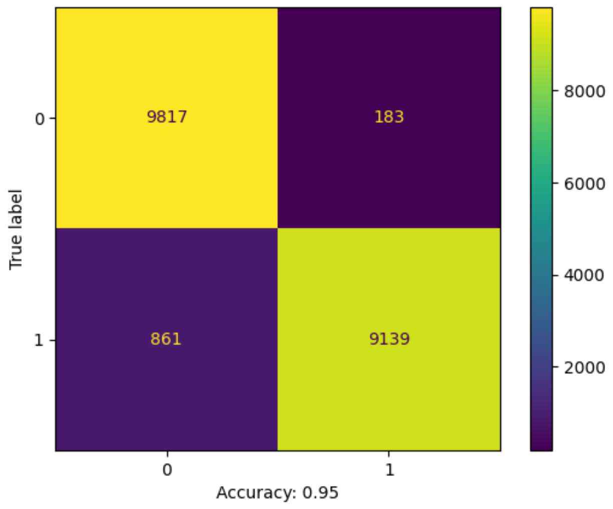

Figure 9: CNN Confusion Matrix

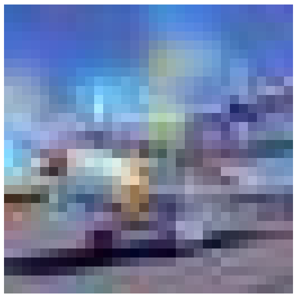

Figure 10: GradCam Image Fake Airplane

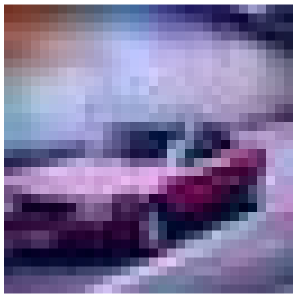

Figure 11: GradCam Image on Fake Car

7 Discussion

The model seems to more frequently predict class 0 (fake) than class 1 (real) which could be due to the slightly higher amount of class 0 images. However, this has the implication of it more frequently having a false negative than a false positive which would be a less harmful result. Furthermore, the extremely high training and testing accuracy scores of 99% and 95% show that the CNN is able to capture the spatial relationships between pixels in an image in a far superior manner than the random forest.

The CNN did not originally perform this well since I started by importing the resnet50 model and then adding dense layers to the end. However, the model was not performing that much better than the random forest, which led me to create a custom CNN and getting full control over each layer. Mirroring the VGG16 architecture led me to the higher performing model along with tweaking around the dropout values (empirical testing).

It would be massively beneficial to continue this research since stable diffusion models and GANs are only becoming more advanced, posing an increasing threat public media and research (assuming these are used maliciously). Also, it would be interesting to see how the model performs with images of larger sizes because the current ones are only 32x32. This causes the resolution to be quite low, which could pose a problem for feature extraction.

Overall, this is a good model as it has a high accuracy, precision, and recall all while not being overfit. Its primary pitfall is predicting class 0 more frequently, even if it is not, which is a safer error to make. It is better to err on the side of caution to prevent the spread of any misinformation.