Final Project

Graham Annett and Jason Shenny

Friday, March 22, 2019

Abstract

For our final project, we chose to use the data analysis skills we learned in our 582 course to classify images that had been generated by deep learning networks from those contained in the training set used to train the deep learning models. Utilizing a variety of techniques we learned in class we showed that with various preprocessing techniques paired with supervised learning methods, there is some predictive power in differentiating between real photos and those generated from complex deep learning models.

---

1 Problem Background

Generative deep learning is an incredibly hot topic that utilizes deep learning architectures to generate images and texts. While these generated texts and images are created from a training dataset and thus exhibit only artifacts of what they have previously seen, many have pointed out that these computer generated items can be used maliciously. Because of this ethical concerns are raised and the desire to distinguish what is computer generated and what is not is becoming a hot topic. There are now even a variety of sites generating fake images intended to show that generation is so good that is can confuse you by making you think the image does actually exist[1]. While the design and architecture of these models is beyond the scope of this paper, a good resource for information can be found at this footnote[2].

While the models used to generate the images are much more complex, computationally expensive and data hungry, we are only interested in evaluating the capability of methods learned in 582 in offering insight into the world of deep learning. One of the issues frequently brought forth throughout this course was that while there may be many advanced and complex models and techniques to approach problems nowadays, it is also hard to know if they are actually doing what is intended. The complex algorithm structure and methodology can obscure the mathematical operations used for simple toy problems. We believe this is a good example of one of those toy problems and we see that our model results exhibit what was anticipated (that our results are not much better than guessing). Techniques and methodologies that do not work or translate to harder problems is an important aspect of scientific research as well. However, the use of these simpler techniques and methodologies can offer advantageous when performed selectively and properly.

2 Theory and Background

Preprocessing techniques such as the Fast Fourier Transform (FFT) and Singular Value Decomposition (SVD) are at the core of machine learning studies and data classification problems. The mathematical operations that define these techniques can make or break a machine learning algorithm. The goal of them is to filter negligible (FFT) or redundant (SVD) information in the data by transforming the original 'coordinate system' or basis vectors to one that is more appropriate for describing the similarities and differences in the data.

The FFT can be used to 'prime' the data prior to feeding into the SVD. This priming is the transformation from the original (in the case spatial) coordinates to frequency coordinates. From their the SVD can rotate and stretch the data to align in a fashion that collapses redundant information (highly covariant data) and highlights the areas of high variance. An interesting side note regarding this process is that SVD is a ubiquitous method for dimensionality reduction, whereas the FFT is one transformation type amongst many different types.

It may very well be that the FFT is not the best method to prime the data prior to SVD but in the cases of spatial image structure we know from experience that it is helpful. As seen in **Figure 1** and **Figure 2**, the FFT'd images are summed up in a 30x30 pixel box, whereas the images themselves are 128x128.

[1] https://thispersondoesnotexist.com
[2] https://www.lyrn.ai/2018/12/26/a-style-based-generator-architecture-for-generative-adversarial-networks/

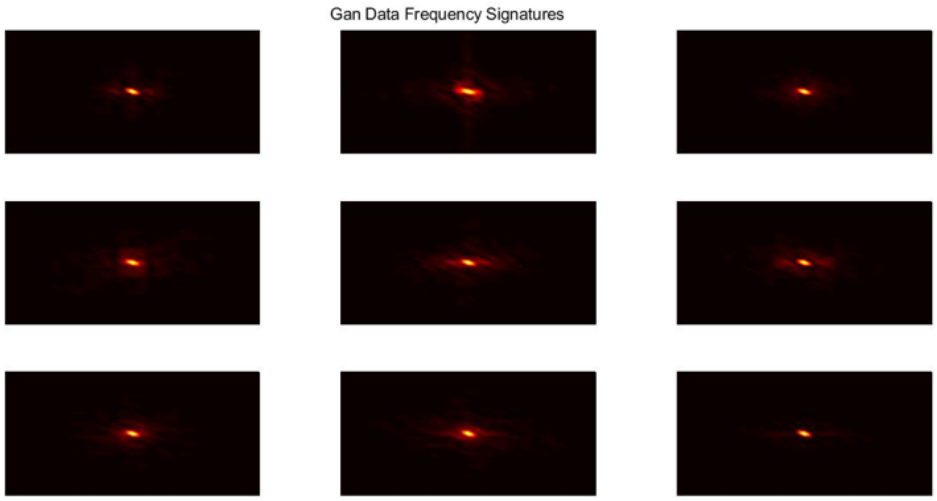

Figure 1: GAN Frequency Spectrum Visualization

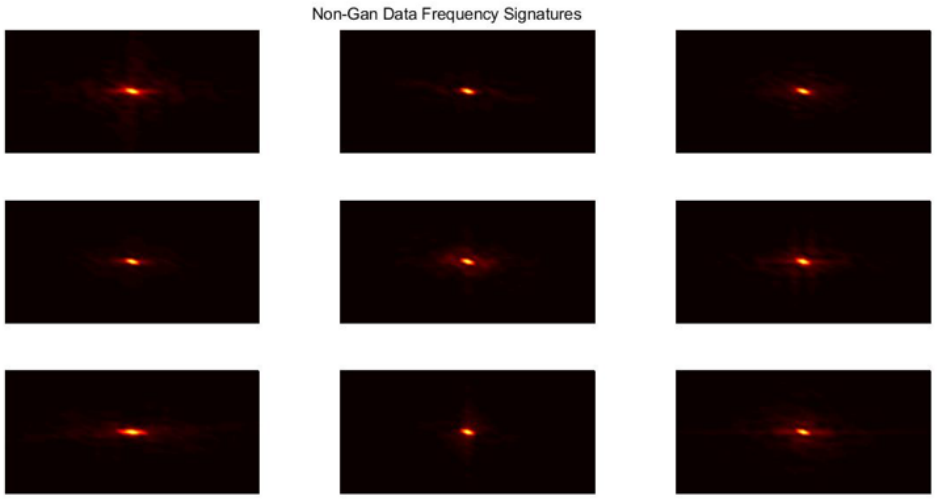

Figure 2: Non-GAN Frequency Spectrum Visualization

From here, the rest of that 128X128 can be filtered down to zero, compressing the data. Classification between GAN and non-GAN frequency signatures is very difficult to be performed with just the human cognition, especially with large numbers of images. Plots of the frequency properties are shown below in **Figure 3** and **Figure 4** to remove the need for visualization and memory.

The maximum amplitude frequency for all GAN and Non-GAN images occur directly in the center of the frequency spectrum at and their maximum amplitude is all on the same order of magnitude of 10^6. This method of analysis also proves to be inadequate in discerning a non-GAN image from a GAN image. This is where we turn to other computation methods.

The FFT can be thought of as transforming the structure of the data. The operator behind this transformation is always performed the exact same way regardless of the properties of the dataset. Because of this the FFT, can be thought of as dummy operation, however useful when performed on the appropriate dataset. The SVD, however; is smart. The SVD can be thought of as shifting the data's structure to the most optimal, orthogonal basis vectors for the unique properties contained within the dataset. This shifting procedure is a function of rotation and stretching/compression. The SVD operator decides what direction the rotation and stretch is performed to align the data to its naturally

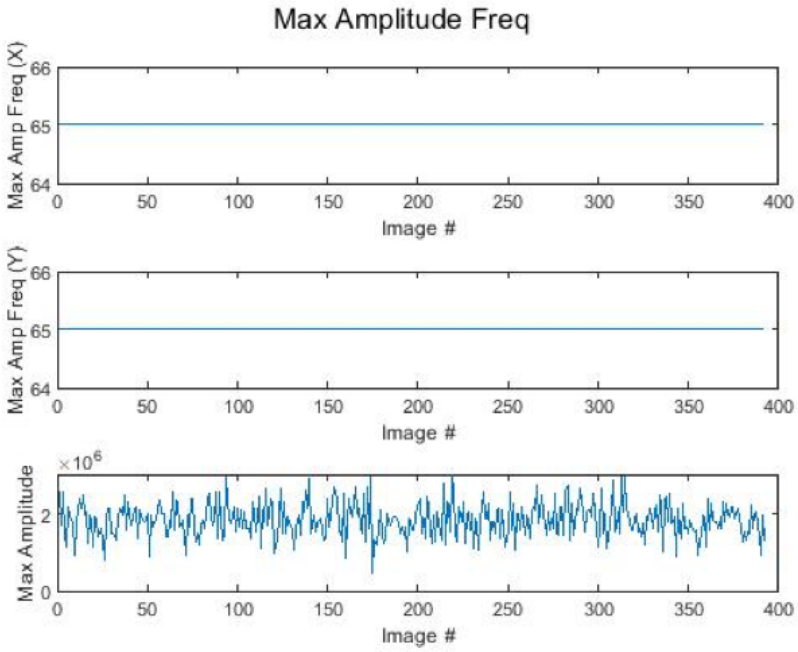

Figure 3: GAN Frequency Properties

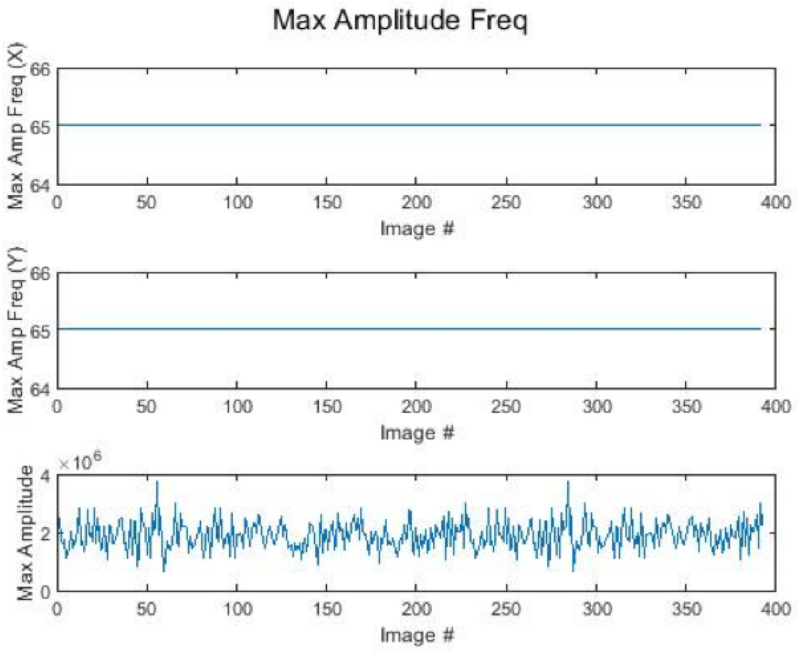

Figure 4: Non-GAN Frequency Properties

most important directions of activity. These new vectors that come out are ranked based on this importance through a variance calculation. From there the unimportant ones can be truncated. This is another form of compression, but through dimensionality reduction. **Figures 3** and **Figure 4** show plots of the singular value spectrums which detail the energy contained in each SVD mode for both the GAN and Non-GAN data.

The modes that come out of the SVD are independent and orthogonal directions in the data. For images, these modes are commonly called eigenfaces and they describe common facial features within the data. For this GAN non-

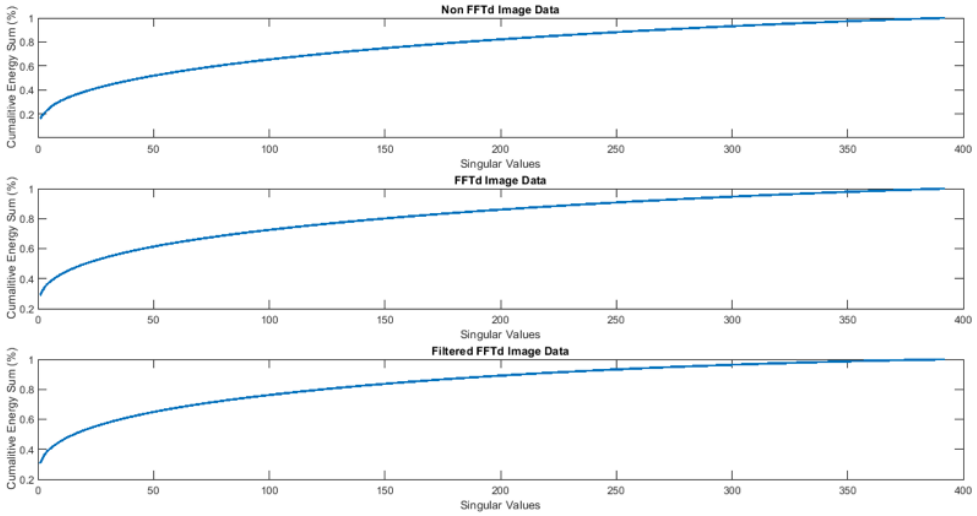

Figure 5: GAN Singular Value Spectrums

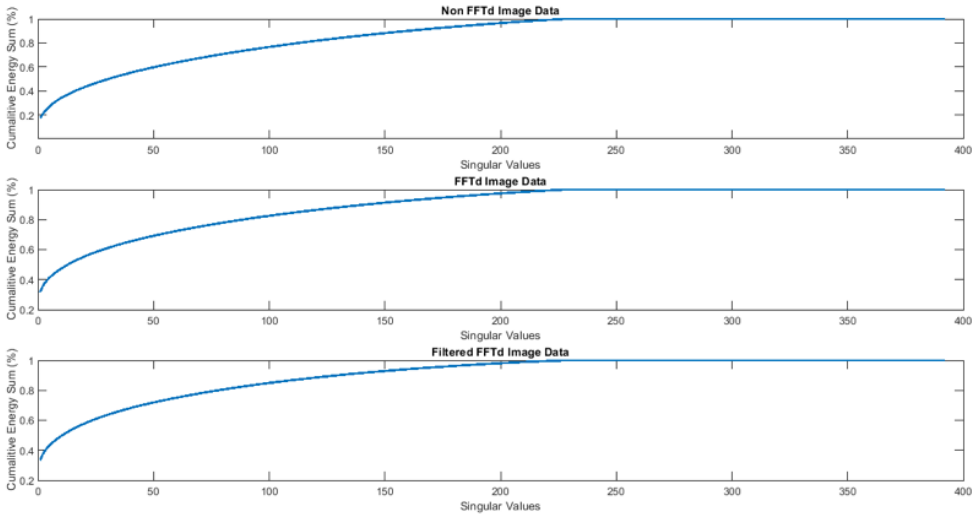

Figure 6: Non-GAN Singular Value Spectrums

GAN classification project, this could have proved to be very useful if the GAN data contained quantitative differences in either the spatial or frequency domain. Based on the findings within this report, it did not and the small, observable imperfections in the GAN data proved to be hidden to these preprocessing methods.

Orthogonality is a very advantageous quality of the SVD but also has its fall backs. Because the SVD modes must be orthogonal, the SVD operation is blind to data that would be optimally separated by two nonorthogonal basis' vectors. Cases such as images with reflections where there two or multiple independent entities are colliding on the same data set get treated as one. Here another method, Independent Component Analysis (ICA), yields advantageous. ICA is not needed for this assignment because the images used are not meant to be obscured with reflections and the imperfections in the GAN data are inherent in the single stream structure of the image. The machine learning algorithms used herein explore the advantages of various combinations of FFT and SVD preprocessing techniques. The preprocessing can be adjusted to suit the desired efficiency / accuracy trade off.

[3] https://github.com/NVlabs/stylegan
[4] https://github.com/NVlabs/ffhq-dataset

Figure 7: Example of Both Training and GAN Images

3 Algorithm

3.1 Preprocessing

The datasets we use come from the training dataset and generated images provided here[3] which is the official repo for the StyleGAN paper. These datasets are quite large with the original images they used from a Flickr dataset[4] being 955GB. They also offer a subset of images that are 1024x1024 cropped of the faces which is what the StyleGAN network was trained on for the faces model. We are also using part of the 100,000 generated faces dataset they provide that are generated from the Flickr dataset.

With the data we are using, we realistically cannot make any assumptions about the distributions of fake to real (in some future dystopia, it could be the anomaly to find real images) and as such we will need to have equal proportions of both. This is important as otherwise our classification rate may give us better accuracy than we may actually have just due to the data being highly skewed for a specific type of image (say real images, where there are likely many more online at this point in time than the images generated via deep learning).

Along with making sure we are capturing equal distributions of the data, we are using cross-validation to split the data into 10 separate train-test splits where 90% of the data is used for training and 10% is withheld from all data analysis techniques until we have built the model. Looking through the datasets as in **Figure 7**, the photos fall on a spectrum for both realistic photos (i.e. many real photos have interesting angles such as the 1st image on the top row or realistic flaws such as the small bit of another persons head creeping into the photo on the 4th image on top) as well as the "fake" generated dataset on the bottom row (where you can notice some background and teeth anomalies on the various images). To the best of our knowledge, the real images do not contain any duplicates of people but this is hard to verify.

From here, we take the image and resize to an image size that we are able to computationally handle on our personal machines. With this smaller image, we take the 2-D Fourier transform (via `fft2` in Matlab) of the image. At this point, we split the data into our training and test sets as described above.

With our training set, we compute the SVD of the training images, where each row signifies an image (since this makes more sense to have each row be a sample and thus our label vector will be a column vector). With this SVD, we can project back onto our image space with the first r modes via matrix multiplication:

$$\tilde{\mathbf{V}}_\mathbf{r}\tilde{\mathbf{V}}_\mathbf{r}^*$$

(similar to how for the eigenfaces example, where each face was a column, we projected back with:

$$\tilde{\mathbf{x}}_\mathbf{test} = \tilde{\mathbf{U}}_\mathbf{r}\tilde{\mathbf{U}}_\mathbf{r}^*\tilde{\mathbf{x}}_\mathbf{test}$$

While rank truncation proved valuable for speeding up our model creation and programming iteration, we found our predictive capabilities decreased when using a limited set of modes for the few model types that provided possible predictive power. This makes sense given that possible deviations from modes containing small amounts of the images "energy" is what could be a good indicator for what makes an image real or not.

3.2 Naive Bayes

The Naive Bayes classifier is model type that is based around Bayes theorem and gives us the ability to look at our model in the framework of a conditional probability. Because of this, it does not always extrapolate well to datasets with a very large number of features[5]. This can be a problem for datasets related to images (where the feature space will be $pixel\_width \times pixel\_hight$) when looking at the full feature space of an image. Fortunately, we can look at a low rank truncation of the feature space if we are using a dimensionality reduction technique such as the SVD to mitigate this issue for model types such as the Naive Bayes classifier.

3.3 Support Vector Machines

Support Vector Machines or SVM, is a classification and regression method that allows the data to be fit via some hyperplane that best separates the data. SVMs have been used for many different forms of classification and have been shown to even perform well in many types of image classification long before the recent buzz around deep learning[6].

3.4 Classification Trees

Classification Trees are based on the idea that you can create a model based on a decision tree that branches based on a features values. While they can be quite intuitive to understand, they often are criticized for being prone to overfitting. To mitigate this, tree based models can be pruned and tuned with various other techniques.

3.5 Adaboost

The AdaBoost model is a form of ensemble learning and for us utilized a tree learner. These tree learners are very similar in many regards to Random Forest models but allow the model to be more sensitive to noisy data and outliers[7]. AdaBoost has also done surprisingly well in online Machine Learning competitions on sites such as Kaggle.com and generalizes to many different Data Science problems.

3.6 Unsupervised Learning

We wanted to inspect two different forms of unsupervised learning to see if there was any natural clusters or information we may glean without the use of labels. Unsupervised learning is based on the idea that rather than having a target label or dependent variable used to dictate clusters, a fitting algorithm is iterated to convergence to separate unknown but learned differences in the data. For our data, we found clustering to not entirely make sense based on the notion that we are reforming images into vectors and then using distance metrics to create clusters (on data where specific pixels have no inherent underlying relation). Rather, our intuition was that maybe with our results from spectrum analysis and SVD we would find a way to incorporate an unsupervised learning algorithm with our dataset. Using our SVD outputs, we chose to project back onto the PCA Modes and use K-Means clustering. While this was not going to solve the problem with identifying generated images, we believed that there was a chance that maybe some amount of images would exhibit phenomena that a prominent cluster would form around. With this cluster perhaps, we could identify such a way that further investigation would provide us some insight into what types of images may be easily categorized as real or fake. K-Means clustering allows us to separate the data without labels, into K number of specific centroids.

3.7 Alternative Tests

3.7.1 Test 2

The lady in this image, **Figure 8**, has some weird texture around her lip, almost as if the photo was burned and the background doesn't make sense (the background feels a bit surrealist even). We find the background being off is a common thing in many of these generated images and perhaps this is a specific avenue that we could look to discriminate between real and generated images (i.e. taking the upper corner of the image).

In our computational results, we show the models to have some improvement on our models predictability but not a

[5] From the Naive-Bayes Wikipedia: *The problem with the above formulation is that if the number of features n is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible.* https://en.wikipedia.org/wiki/Naive_Bayes_classifier#Probabilistic_model

[6] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.459.9821&rep=rep1&type=pdf

[7] https://en.wikipedia.org/wiki/AdaBoost

Figure 8: Image "000202" From GAN Dataset

large amount and impossible to ascertain if it would generalize well to all the faces in the real images and generative dataset.

3.7.2 Test 3

For *Test 3*, we did the same process as *Test 1* but chose to forgo using spectral analysis on the images. The idea here is that there is a possibility that classical supervised learning techniques could be used to have some predictive power when discerning real from generated images. This method also worked quite well and actually surpassed both of the methods that incorporated spectrum analysis.

4 Computational Results

4.1 FFT and SVD Results

There is one noticeable disparity apparent in the singular value spectrum visualization. The Filtered FFTd GAN images contain more energy in the first 700 modes than the non-filtered FFT, and non FFTd energy spectrums. This is not the case for the non-GAN image SV spectrum. A comparison of energy spectrums for the non-GAN shows that filtered FFT, full FFT and non FFTd all have very similar mode-energy relationships. Since the filtering was done with the same Boolean filter shown below, this could mean that relevant frequency information was contained in the filtered negligible space for the GAN images that was not inherent in the Non-GAN images. Next we will look at the truncation error as a function of the number of modes used in the reconstruction of the data fed into the SVD. See the reconstruction equation:

$$\mathbf{X_r} = \mathbf{U_r S_r V_r}$$

The comparison of these two figures show an exponentially decaying error when more and more modes are used for image reconstruction. This as well as the magnitude of the error between No FFT, FFT, and Filtered FFT are as expected. One very interesting phenomenon is that the Non-Gan images take far fewer modes to decrease the truncation error to a negligible amount. This can be used to our advantage by truncating the training and test sets to pull out differences in

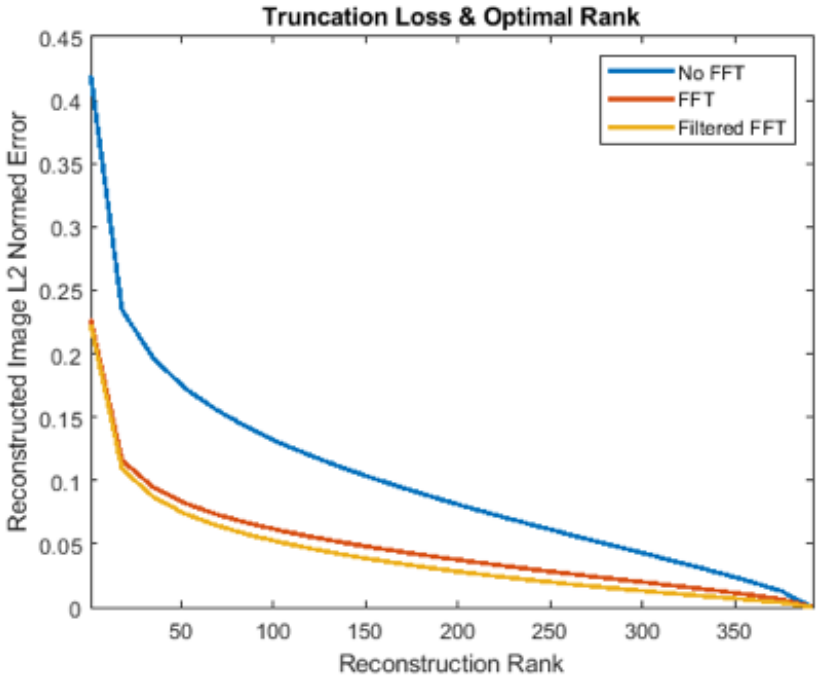

Figure 9: Gan Reconstructed Image Error

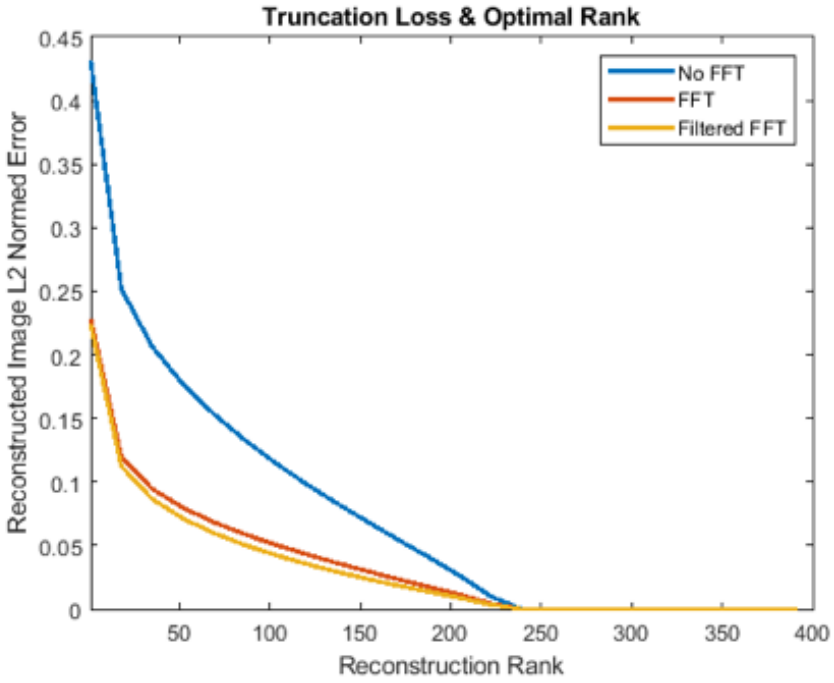

Figure 10: Non-Gan Reconstructed Image Error

4.2 GAN vs Non-GAN data

The idea here is that a truncated version of the SVD projection of a GAN image will not be as clear as a non-GAN image. See above in **Figure 11** and **Figure 12** for the reconstruction of both a GAN and Non-GAN image at various ranks. The reconstruction of non-FFTd data was used with a rank of 250 to optimally highlight the disparity between the impacted GAN images and the non-impacted non-GAN images. The quantitative results of using this truncated reconstruction for classification of GAN vs non-GAN are summarized in the Classification Results table below. Overall this method

Figure 11: GAN Reconstructed Image

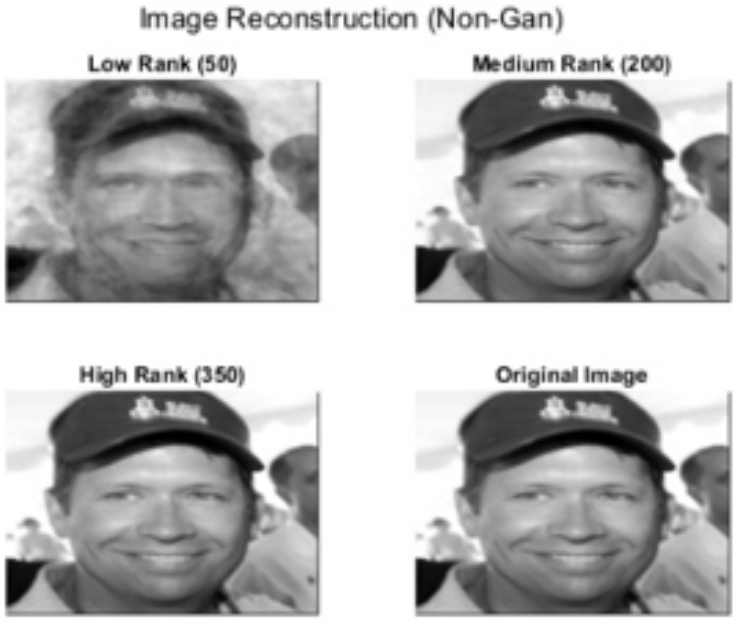

Figure 12: Non-GAN Reconstructed Image

performed very well for the task at hand, showing a strong decrease in test loss from other preprocessing methods. For this test a clear methodology, backed by quantitative reasoning for choosing the data structure and rank truncation was used and the results reflect this.

The two figures above clearly show the dichotomy between GAN and Non-GAN rank truncation and reconstruction. At 250 modes used for reconstruction the Non-GAN image shows a very good reconstruction where as the GAN reconstruction is negatively impacted by the truncation. This a direct quantitative match to the difference in exponential decay

Figure 13: Test 2 - Background Subsample From GAN Dataset

described in **Figure 9** and **Figure 10**.

4.3 Classification Results

While the majority of our results were somewhat close, we wanted to test out a few different hypothesis and see if any would allow for a particular angle or insight we could leverage for better results.

Our first classification test was based around the idea that due to these differences in the images and backgrounds, there was some related underlying color frequency or "texture-related" frequency that we would be able to capture in the image via transforming the image with a Fourier Transform. The idea with this is similar to homework 4 in which we used the *Short Time Fourier Transform* to help us classify our songs by allowing us to look at the songs in the frequency space rather than as just a signal along a time series. By doing this, we thought that perhaps the generated images would exhibit some overarching frequency signal that perhaps the training images did not exhibit.

Along with this initial test case, we then chose to look at only a subsample of the image that would hopefully capture primarily the background or a specific area that might have texture or feature anomalies. For example, as in **Figure 13** we chose to specifically look at the upper left and upper right corners of the images as these portions of the generated dataset seemed to exhibit a large percentage of the overall anomalies we detected and as such hypothesized that these areas may exhibit some particular texture frequency that we could exploit to classify these images.

The test results for both these tests, provided in table below, shows that perhaps there is some predictive capability in some model types such as a Naive-Bayes or AdaBoost, but some of our models such as Classification Decision Tree and SVM are unable to do much better than a random coin flip (given that our underlying distributions of images are equal). Our cross-validation uses a 10 K-Fold Split, meaning that we take the data into 10 distinct splits where for each of these splits, 9/10ths of the data will be used for training and 1/10th of the data will be used for testing.

Classification Results				
Test Number - Test Description Misclassification Value From	Naive-Bayes	SVM	Decision Tree	AdaBoost
Test 1 - Full Image with FFT				
Train Loss	0.3589	0.4967	0.0137	0
CrossValidation K-Fold Loss	0.4701	0.4977	0.4959	0.4208
Test 2 - Subsection with FFT				
Train Loss	0.4466	0.5123	0.0192	0
CrossValidation K-Fold Loss	0.5000	0.4918	0.4396	0.3922
Test 3 - Full Image No FFT				
Train Loss	0.3067	0.0618	0.0172	0
CrossValidation K-Fold Loss	0.3475	0.1663	0.3508	0.1864
Test 4				
Train Loss	0.3017	0.1168	0.0122	0
CrossValidation K-Fold Loss	0.3455	0.2847	0.3966	0.2579

Figure 14: Misclassification From Test 1 - Difficult To Discern "Real"

For some of the models, and particularly for the AdaBoost model, the models not only seem to fare above average, but there does seem to be some amount of model predictability by subsampling the image to be a corner. While the testing for these models and dataset used was quite small compared to the entire datasets, we believe that our models most likely would generalize to the entire dataset and that while these models are by no means great or should be used in production, there is probable chance that further research could generate higher predictability using these methods to differentiate computer generated images from real life images.

For our problem, we believe that in particular the SVM and and Naive Bayes methods were not capable of doing above a random guess because the problem is similar in many regards to anomaly detection amongst a random subsection of each photograph. Because of this, the model types that were tree based such as AdaBoost and Classification Decision Tree were able to overfit to the training data while still maintaining some amount of predictive power. While we did not use any robust statistical measures, we believe anything outside of +/- 5% of 50% misclassification rate suggests

Figure 15: Misclassification From Test 1 - Obvious Which Is "Fake"

further research could provide valuable insight.

When inspecting some photos that were misclassified in *Test 1* such as in **Figure 14**, we found no distinct pattern that we could visually discern what made this specific image seem generated or made the other appear "real". As such we are not surprised our very rudimentary models misclassified them as we have trouble discerning what label belongs to which. On the other hand, as in **Figure 15** sometimes the images are

When looking into some images that were misclassified amongst the multiple different tests, we found some overlaps but the misclassifications were not entirely consistent enough amongst different tests to make sense of without further investigation.

For our unsupervised data exploration, we were unable to get any results using K-Means that visualized well by the nature of the photos using a high amount of pixels. Our hope was that using PCA Projection with K-Means we would be able to see something about a section of our photos or have our photos spread out in a particular manner depending on the modes we chose. Instead what we found is in **Figure 16**. We frequently saw that when using PCA Projection of two modes, our data would tend to randomly cluster for the training images but for the GAN images it tended to lie almost on a perfectly straight slope. We were unable to figure out exactly why this was.

5 Drawbacks and Future Research Avenues

While it is impossible to pinpoint one specific feature that makes classifying the generated from the real images, there are many images where the model generated images have significant anomalies such as in **Figure 8**. One of the things we can notice is that many of the discrepancies between real and fake images seem to be either due to weird textures in the background as well as weird anomalies on some aspects of peoples clothing and skin. While our project looked at grayscale images, we believe that there is a possibility that the models could perform better if generalized to take color image matrices. Part of our belief in this is that the oddities often exhibited are often hard to pinpoint precisely or give a generalized avenue to discriminate real from fake images, while allowing color would give our dataset another dimension in a sense to allow certain model types to classify on. While we are hesitant to say with any certainty it would improve results, it would be incredibly interesting as well to not have to resize the image (instead of a subsample that is small enough for us to SVD on our personal computers) so that possible anomalies in texture or color offer larger

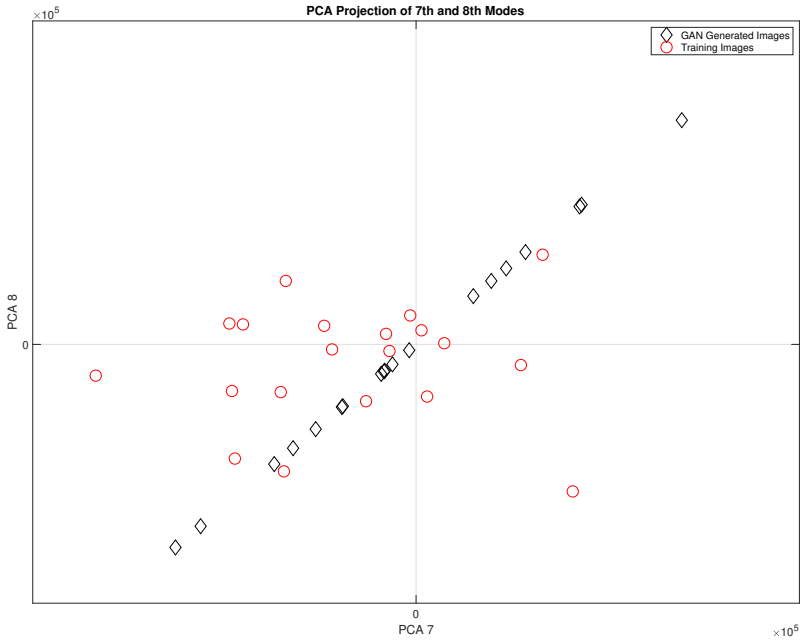

Figure 16: PCA Projection

overall variance and could be used as an avenue to classify images. Along with needing to include more images for this to possibly generalize well with the increased size of the faces as well as dimension, it would also require a lot more compute power than we have personally available to us as all of our data matrices would be triple the size at a minimum and grow incredibly quick as we increase the size of the images as well.

One of the final drawbacks to note is that while these computer generated images are possible to differentiate by humans, research allowing a novel method to differentiate them could be quickly surpassed if these types of differentiating were incorporated into the GAN models discriminative network loss function. Because of this and the nature of these sorts of models continually being able to improve on their ability to generate and create newer more realistic outputs with better model architecture, more compute time and more data, it is likely that any edge that may be gleaned in the short term from many of these more traditional data analysis techniques will in the long-run prove fruitless or be easily integrated into new model architecture types.

6 Conclusion

The use of data processing (FFT), basis vector transformation (SVD), and machine learning techniques learned in AMATH 582 have proven to be successful in distinguishing between real photos of people's faces and images created by Generative Adversarial Networks (GAN). This report explores the advantages and results of feeding multiple machine learning algorithms with various preprocessing techniques. In conclusion Test 3 and Test 4 scored the highest on all machine learning models used. It is most likely not by coincidence that both tests did not take the FFT in the preprocessing. Their better results relative to the other methods could be due to the FFT washing out or hiding the imperfections in the non-GAN data. This is a very interesting finding as it is slightly counterintuitive. We use the FFT to put the data in a better coordinate system. But in this case it may do the opposite by transforming to the frequency spectrum and washing out differences in the data. Maybe there is a transformation to another coordinate system that would be more advantageous for pre-SVD processing but that is beyond the scope of this report. In conclusion, next time you have a feeling that an image you are looking at is not a real photo and you want to find out, it is possible using FFT will not add anything. Do a rank truncated SVD reconstruction and run it through an AdaBoost machine learning model for the most straight forward results.

7 Code Appendix

7.1 Functions Used for Data

```
classdef f

    properties
        folderpath
        all_files

        images_matrix
        images_fftd
    end

    methods
        function obj = f(folderpath)
            %DATASET Construct an instance of this class
            obj.folderpath = folderpath;
        end

        function obj = get_matrix(obj,inputArg)
            %METHOD1 Summary of this method goes here
            %   Detailed explanation goes here
            obj.images_fft = inputArg;
        end
    end

    methods(Static)

        function alldata = combine_data(data1, data2)

            data1labels = ones(size(data1, 1), 1);
            data2labels = ones(size(data2, 1), 1)*2;

            data1 = [data1, data1labels];
            data2 = [data2, data2labels];

            alldata = [data1; data2];
            % data1 and data2 are matrices

        end



        function all_images = images_to_matrix(images_struct, resize_dim)
            all_images = {};

            for k=1:numel(images_struct)
                curr_image = imread(fullfile(images_struct(k).folder,
                    images_struct(k).name));

                % grayscale and resize
                curr_image = rgb2gray(curr_image);
                curr_image = imresize(curr_image, [resize_dim, resize_dim]);
                curr_image = double(curr_image);
                all_images{k} = reshape(curr_image, [], 1);
```

```
        end

        all_images = cell2mat(all_images);
    end

    function all_images = corner_of_images_from_matrix(image_matrix,
        corners_idx, im_m)
        corner_down = corners_idx(1);
        corner_right = corners_idx(2);

        all_images = [];
        for k=1:size(image_matrix, 2)
            curr_image = reshape(image_matrix(:, k), im_m, im_m);
            curr_image = curr_image(1:corner_down, 1:corner_right);
            curr_image = reshape(curr_image, [], 1);
            all_images = [all_images curr_image];
        end
    end


    function [images_fft, mx, imax] = fft_matrix(images_matrix, im_m)
        % FFT + relevant data index loop
        for k = 1:size(images_matrix,2)
            tmp_image = reshape(images_matrix(:,k), im_m, im_m);
            tmp_image = fft2(tmp_image);
            tmp_image = fftshift(abs(tmp_image));
            images_fft(:, k) = reshape(tmp_image, [], 1);

            [m, im] = max(images_fft(:,k));
            mx(k) = m;
            imax(k) = im;
        end
    end

    function all_files = read_folder(varargin)
        file_types = {'*.jpg', '*.png'};
        if nargin == 0
            folderpath = '.';
        elseif nargin == 1
            folderpath = varargin{1};
        else
            error('Too many input arguments.')
        end

        all_files= [];
        for file_type = file_types

            listing = dir(fullfile(folderpath, file_type{:}));
            if 0 < length(listing)
                all_files = [all_files; listing];
            end
        end
    end

    function all_files = read_subfolders(foldername)
        listing = dir(foldername);
```

```
107
108             dirFlags = [listing.isdir] & ...
109                 ~strcmp({listing.name},'.') & ~strcmp({listing.name},'..');
110             subdirs = listing(dirFlags);
111
112             all_files = [];
113             for subdir=subdirs'
114                 subdir_files = f.read_folder(fullfile(subdir.folder, subdir.
                        name));
115                 all_files = [all_files; subdir_files];
116             end
117         end
118
119         function curr_image = read_single_image(data, k)
120             curr_image = imread(fullfile(data(k).folder, data(k).name));
121         end
122
123
124
125     end
126
127
128 end
```

7.2 Test 1

```
1  %% Notes
2
3
4  %% clearing section
5  clear all;
6  clc;
7  close all;
8
9  %% vars
10
11 im_m = 128;
12 im_mm = [im_m, im_m]; % for simple reshape
13 im_size = im_m*im_m;
14
15
16 %%
17 models(1).model_func = @fitcnb;
18 models(1).name = 'Naive Bayes';
19 models(1).test_loss = [];
20 models(1).train_loss = [];
21
22 models(2).model_func = @fitcecoc;
23 models(2).name = 'Multiclass SVM';
24 models(2).test_loss = [];
25 models(2).train_loss = [];
26
27 models(3).model_func = @fitctree;
28 models(3).name = 'Classification Decision Tree';
29 models(3).test_loss = [];
```

```
models(3).train_loss = [];

models(4).model_func = @fitcensemble;
models(4).name = 'Ada Boost';
models(4).test_loss = [];
models(4).train_loss = [];

%% data init section

gan_image_filepath = 'data/gan_generated';
nongan_image_filepath = 'data/non_gan';


% Section for GAN Images
gan_data = f.read_subfolders(gan_image_filepath);
gan_matrix = f.images_to_matrix(gan_data, im_m);
[gan_fft, g_mx, g_imax] = f.fft_matrix(gan_matrix, im_m);


% Section for non-GAN Images
ngan_data = f.read_subfolders(nongan_image_filepath);
ngan_matrix = f.images_to_matrix(ngan_data, im_m);
[ngan_fft, ng_mx, ng_imax] = f.fft_matrix(ngan_matrix, im_m);



% allow for equal samples of training and gan images
num_ngan = size(ngan_fft, 2);
gan_fft_not_included = gan_fft(:, num_ngan:end);
gan_fft = gan_fft(:, 1:num_ngan);

%% train test split - (test split not in svd)
disp('creating cv partition')
cv = cvpartition(size(gan_fft, 2) + size(ngan_fft, 2),'KFold',10);

% each row is a song
all_data = f.combine_data(gan_fft', ngan_fft');

%%


for k=1:cv.NumTestSets
    disp('cv split loop')
    train_data_fft = all_data(cv.training(k), :);

    % each row is a pic, last row is labels
    [u, s, v] = svd(train_data_fft(:, 1:end-1), 'econ');

    % split labels and project back on to test_data
    test_data_fft = all_data(cv.test(k), :);
    test_data_features = test_data_fft(:, 1:end-1);
    test_data_labels = test_data_fft(:, end);
    test_data_features_projected = test_data_features * (v * v');

    % for simplicity in understanding
    trainingX = train_data_fft(:, 1:end-1) * (v * v');
```

```
    trainingY = train_data_fft(:, end);

    for m=1:numel(models)
        mdl = models(m).model_func(trainingX, trainingY);
        models(m).mdl{k} = mdl;

        % calculate train and test loss
        models(m).train_loss = [models(m).train_loss; ...
            loss(mdl, trainingX, trainingY)];

        models(m).test_loss = [models(m).test_loss; ...
            loss(mdl, test_data_features_projected, test_data_labels)];
    end
end

%% calculate cv stuff for first model of each type
disp('doing crossfold')
% for k=1:length(models)
%     mdl = models(k).mdl{1};
%     crossval_mdl = crossval(mdl);
%     cvloss = kfoldLoss(crossval_mdl);
%     models(k).crossval_mdl = crossval_mdl;
%     models(k).cvloss = cvloss;
% end

%% calculate crossval score with all_data svd

[ADu, ADs, ADv] = svd(all_data(:, 1:end-1), 'econ');
ADtrainingX = all_data(:, 1:end-1) * (ADv * ADv');
ADtrainingY = all_data(:, end);
cvmodel = fitcensemble(ADtrainingX, ADtrainingY, 'CrossVal', 'on');
classError = kfoldLoss(cvmodel);


%% unsupervised learning - PCA projection



N_ppl = 20;
gan_fft_pca = gan_fft(:, 1:N_ppl)';
ngan_fft_pca = ngan_fft(:, 1:N_ppl)';


PCAModes = [7 8];
PCACoordsG = gan_fft_pca*v(:,PCAModes);
PCACoordsNG = ngan_fft_pca*v(:,PCAModes);

figure(1)
plot(PCACoordsG(:,2),PCACoordsG(:,2),'kd', 'MarkerSize', 14)
set(gca,'XTick', 0, 'YTick', 0);
hold on, grid on
plot(PCACoordsNG(:,1),PCACoordsNG(:,2),'ro', 'MarkerSize', 14)
xlabel('PCA 7'), ylabel('PCA 8')
legend('GAN Generated Images', 'Training Images')
set(gca,'XTick', [0], 'YTick', [0]);
title('PCA Projection of 7th and 8th Modes')
```

```
set(gca, 'FontSize', 14)

%% visualization

figure(2)
s_diags = diag(s);
plot(s_diags(1:100)/sum(s_diags), 'ko')


%% viz2

figure(3)
for k=1:length(models)
    plot(models(k).test_loss), legend(models(k).name), hold on
end
legend({models.name})



%% vizualation 3

figure(4)
mdl = models(4).mdl{1};
all1s = (gan_fft' * v * v');
all2s = (ngan_fft' * v * v');

preds1s = predict(mdl, all1s);
preds2s = predict(mdl, all2s);

mis1_as2 = randsample(find(preds1s == 2), 1);
mis2_as1 = randsample(find(preds2s == 1), 1);

% 98, 208 are good.
% mis1_as2 = 136;
% mis2_as1 = 45;
subplot(2,2,1)
imshow(reshape(uint8(gan_matrix(:, mis1_as2)), im_m, im_m))
title('Generated Image Misclassified as Real')
set(gca, 'Fontsize', 14)

subplot(2,2,2)
orig_image = f.read_single_image(gan_data, mis1_as2);
imshow(orig_image)
title(['Original Image=', num2str(mis1_as2)])
set(gca, 'Fontsize', 14)


subplot(2,2,3)
imshow(reshape(uint8(ngan_matrix(:, mis2_as1)), im_m, im_m))
title('Real Image Misclassified as Generated')
set(gca, 'Fontsize', 14)

subplot(2,2,4)
orig_image = f.read_single_image(ngan_data, mis2_as1);
imshow(orig_image)
title(['Original Image=', num2str(mis2_as1)])
```

```
set(gca, 'Fontsize', 14)
```

7.3 Test 2

```
%% Notes


%% clearing section
% clear all;
clc;
close all;

%% vars
disp('running noftt_test')

im_m = 128;
im_mm = [im_m, im_m]; % for simple reshape
im_size = im_m*im_m;


%%
models(1).model_func = @fitcnb;
models(1).name = 'Naive Bayes';
models(1).test_loss = [];
models(1).train_loss = [];

models(2).model_func = @fitcecoc;
models(2).name = 'Multiclass SVM';
models(2).test_loss = [];
models(2).train_loss = [];

models(3).model_func = @fitctree;
models(3).name = 'Classification Decision Tree';
models(3).test_loss = [];
models(3).train_loss = [];

models(4).model_func = @fitcensemble;
models(4).name = 'Ada Boost';
models(4).test_loss = [];
models(4).train_loss = [];

%% data init section

gan_image_filepath = 'data/gan_generated';
nongan_image_filepath = 'data/non_gan';


% Section for GAN Images
gan_data = f.read_subfolders(gan_image_filepath);
gan_matrix = f.images_to_matrix(gan_data, im_m);


% Section for non-GAN Images
ngan_data = f.read_subfolders(nongan_image_filepath);
ngan_matrix = f.images_to_matrix(ngan_data, im_m);
```

```
52
53  % allow for equal samples of training and gan images
54  num_ngan = size(ngan_matrix, 2);
55  gan_matrix_not_included = gan_matrix(:, num_ngan:end);
56  gan_matrix = gan_matrix(:, 1:num_ngan);
57
58  %% train test split - (test split not in svd)
59  disp('creating cv partition')
60  cv = cvpartition(size(gan_matrix, 2) + size(ngan_matrix, 2),'KFold',10);
61
62  % each row is a song
63  all_data = f.combine_data(gan_matrix', ngan_matrix');
64
65  %%
66
67  for k=1:cv.NumTestSets
68      disp('cv split loop')
69      train_data_ = all_data(cv.training(k), :);
70
71      % each row is a pic, last row is labels
72      [u, s, v] = svd(train_data_(:, 1:end-1), 'econ');
73
74      % split labels and project back on to test_data
75      test_data_ = all_data(cv.test(k), :);
76      test_data_features = test_data_(:, 1:end-1);
77      test_data_labels = test_data_(:, end);
78      test_data_features_projected = test_data_features * (v * v');
79
80      % for simplicity in understanding
81      trainingX = train_data_(:, 1:end-1) * (v * v');
82      trainingY = train_data_(:, end);
83
84      for m=1:numel(models)
85          mdl = models(m).model_func(trainingX, trainingY);
86          models(m).mdl{k} = mdl;
87
88          % calculate train and test loss
89          models(m).train_loss = [models(m).train_loss; ...
90              loss(mdl, trainingX, trainingY)];
91
92          models(m).test_loss = [models(m).test_loss; ...
93              loss(mdl, test_data_features_projected, test_data_labels)];
94      end
95  end
96
97  %% calculate cv stuff for first model of each type
98  disp('doing cross val on single model')
99  for k=1:length(models)
100     mdl = models(k).mdl{1};
101     crossval_mdl = crossval(mdl);
102     cvloss = kfoldLoss(crossval_mdl);
103     models(k).crossval_mdl = crossval_mdl;
104     models(k).cvloss = cvloss;
105 end
106
107 %% calculate crossval score with all_data svd
```

```

[ADu, ADs, ADv] = svd(all_data(:, 1:end-1), 'econ');
ADtrainingX = all_data(:, 1:end-1) * (ADv * ADv');
ADtrainingY = all_data(:, end);
cvmodel = fitcensemble(ADtrainingX, ADtrainingY, 'CrossVal', 'on');
classError = kfoldLoss(cvmodel);


%% unsupervised learning - PCA projection

N_ppl = 20;
gan_pca = gan_matrix(:, 1:N_ppl)';
ngan_pca = ngan_matrix(:, 1:N_ppl)';


PCAModes = [2 3];
PCACoordsG = gan_pca*v(:,PCAModes);
PCACoordsNG = ngan_pca*v(:,PCAModes);


figure(1)
plot(PCACoordsG(:,2),PCACoordsG(:,2),'kd')
set(gca,'XTick', 0, 'YTick', 0);
hold on, grid on
plot(PCACoordsNG(:,1),PCACoordsNG(:,2),'ro')
legend('GAN Generated Images', 'Training Images')
set(gca,'XTick', [0], 'YTick', [0]);
% xlabel('PCA Coordinates'), ylabel('

%% visualization

figure(2)
s_diags = diag(s);
plot(s_diags(1:100)/sum(s_diags), 'ko')


%% viz2

figure(3)
for k=1:length(models)
    plot(models(k).test_loss), legend(models(k).name), hold on
end
legend({models.name})
```

7.4 Test 3

```
%% Notes


%% clearing section
clear all; clc;
close all;

%% vars

```

```
im_m = 512;
subsection_idx = [100, 100];

im_mm = [im_m, im_m]; % for simple reshape
im_size = im_m*im_m;


%%
models(1).model_func = @fitcnb;
models(1).name = 'Naive Bayes';
models(1).test_loss = [];
models(1).train_loss = [];

models(2).model_func = @fitcecoc;
models(2).name = 'Multiclass SVM';
models(2).test_loss = [];
models(2).train_loss = [];

models(3).model_func = @fitctree;
models(3).name = 'Classification Decision Tree';
models(3).test_loss = [];
models(3).train_loss = [];

models(4).model_func = @fitcensemble;
models(4).name = 'Ada Boost';
models(4).test_loss = [];
models(4).train_loss = [];

%% data init section

gan_image_filepath = 'data/gan_generated';
nongan_image_filepath = 'data/non_gan';


% Section for GAN Images
gan_data = f.read_subfolders(gan_image_filepath);
gan_matrix = f.images_to_matrix(gan_data, im_m);


% Section for non-GAN Images
ngan_data = f.read_subfolders(nongan_image_filepath);
ngan_matrix = f.images_to_matrix(ngan_data, im_m);



gan_up_right = f.corner_of_images_from_matrix(gan_matrix, subsection_idx, im_m)
    ;
ngan_up_right = f.corner_of_images_from_matrix(ngan_matrix, subsection_idx,
    im_m);

[gan_fft, g_mx, g_imax] = f.fft_matrix(gan_up_right, subsection_idx(1));
[ngan_fft, ng_mx, ng_imax] = f.fft_matrix(ngan_up_right, subsection_idx(1));

% allow for equal samples of training and gan images
num_ngan = size(ngan_fft, 2);
gan_fft_not_included = gan_fft(:, num_ngan:end);
```

```
gan_fft = gan_fft(:, 1:num_ngan);

%% train test split - (test split not in svd)
disp('creating cv partition')
cv = cvpartition(size(gan_fft, 2) + size(ngan_fft, 2),'KFold',5);

% each row is a song
all_data = f.combine_data(gan_fft', ngan_fft');

%%

disp('cv split loop')
for k=1:cv.NumTestSets
    train_data_fft = all_data(cv.training(k), :);

    % each row is a pic, last row is labels
    [u, s, v] = svd(train_data_fft(:, 1:end-1), 'econ');

    % split labels and project back on to test_data
    test_data_fft = all_data(cv.test(k), :);
    test_data_features = test_data_fft(:, 1:end-1);
    test_data_labels = test_data_fft(:, end);
    test_data_features_projected = test_data_features * (v * v');

    % for simplicity in understanding
    trainingX = train_data_fft(:, 1:end-1) * (v * v');
    trainingY = train_data_fft(:, end);

    for m=1:numel(models)
        mdl = models(m).model_func(trainingX, trainingY);
        models(m).mdl{k} = mdl;

        % calculate train and test loss
        models(m).train_loss = [models(m).train_loss; ...
            loss(mdl, trainingX, trainingY)];

        models(m).test_loss = [models(m).test_loss; ...
            loss(mdl, test_data_features_projected, test_data_labels)];
    end
end

%% calculate cv stuff for first model of each type
for k=1:length(models)
    mdl = models(k).mdl{2};
    crossval_mdl = crossval(mdl);
    cvloss = kfoldLoss(crossval_mdl);
    models(k).crossval_mdl = crossval_mdl;
    models(k).cvloss = cvloss;
end



%% unsupervised learning - PCA projection

N_ppl = 20;
gan_fft_pca = gan_fft(:, 1:N_ppl)';
```

```
ngan_fft_pca = ngan_fft(:, 1:N_ppl)';


PCAModes = [2 3];
PCACoordsG = gan_fft_pca*v(:,PCAModes);
PCACoordsNG = ngan_fft_pca*v(:,PCAModes);


figure(1)
plot(PCACoordsG(:,2),PCACoordsG(:,2),'kd')
set(gca,'XTick', 0, 'YTick', 0);
hold on, grid on
plot(PCACoordsNG(:,1),PCACoordsNG(:,2),'ro')
legend('GAN Generated Images', 'Training Images')
set(gca,'XTick', [0], 'YTick', [0]);
% xlabel('PCA Coordinates'), ylabel('

%% visualization

figure(2)
s_diags = diag(s);
plot(s_diags(1:100)/sum(s_diags), 'ko')


%% viz2

figure(3)
for k=1:length(models)
    plot(models(k).test_loss), legend(models(k).name), hold on
end
legend({models.name})
% model_names=
```

7.5 Visualization Tools

```

%% vars

im_m = 512;
im_mm = [im_m, im_m]; % for simple reshape
im_size = im_m*im_m;


%%

gan_image_filepath = 'data/gan_generated';
nongan_image_filepath = 'data/non_gan';

% Section for GAN Images
gan_data = f.read_subfolders(gan_image_filepath);
gan_matrix = f.images_to_matrix(gan_data, im_m);
[gan_fft, g_mx, g_imax] = f.fft_matrix(gan_matrix, im_m);
number_of_gan_images = numel(gan_data);


```

```
21
22 % Section for non-GAN Images
23 ngan_data = f.read_subfolders(nongan_image_filepath);
24 ngan_matrix = f.images_to_matrix(ngan_data, im_m);
25 [ngan_fft, ng_mx, ng_imax] = f.fft_matrix(ngan_matrix, im_m);
26 number_of_ngan_images = numel(ngan_data);
27
28
29 gan_up_right = f.corner_of_images_from_matrix(gan_matrix, [100, 100], im_m);
30 ngan_up_right = f.corner_of_images_from_matrix(ngan_matrix, [100, 100], im_m);
31 %% visualizations
32 % ------------------------------------------------------------------------
33 % notes:
34
35 %%
36
37
38 %% visualization
39
40 % 4 gan images
41 gan_samples = randsample(numel(gan_data), 4);
42 ngan_samples = randsample(numel(ngan_data), 4);
43
44 multi_image_gan = [];
45 multi_image_ngan = [];
46
47 for k=1:length(gan_samples)
48     gan_sample = read_single_image(gan_data, gan_samples(k));
49     gan_sample = imresize(gan_sample, im_mm);
50
51     ngan_sample = read_single_image(ngan_data, ngan_samples(k));
52     ngan_sample = imresize(ngan_sample, im_mm);
53
54
55     % ngan and gan image
56     multi_image_gan = [multi_image_gan gan_sample];
57     multi_image_ngan = [multi_image_ngan ngan_sample];
58 end
59
60
61 figure(1)
62 subplot(2,1,1)
63 imshow(multi_image_ngan)
64 title('Images Generative Adversarial Network Was Trained On')
65 set(gca, 'Fontsize', 14)
66 subplot(2,1,2)
67 imshow(multi_image_gan)
68 title('Images Generated Via Generative Adversarial Network')
69 set(gca, 'Fontsize', 14)
70
71
72 %% showing upper corner image
73 figure(2)
74 n = 15;
75 im1 = uint8(reshape(gan_up_right(:, n), 100, 100));
76 im2 = uint8(reshape(gan_matrix(:, n), im_m, im_m));
```

```
subplot(1,2,1)
imshow(im2), hold on
rectangle('Position',[1,1,100,100],...
    'LineWidth',3,'LineStyle','-', 'Edgecolor', 'r')
title('Original Image')
set(gca, 'Fontsize', 16)
subplot(1,2,2)
imshow(im1)
title('Background Subsection to Analyze')
set(gca, 'Fontsize', 16)

figure(3)
n = 15;
im1 = uint8(reshape(ngan_up_right(:, n), 100, 100));
im2 = uint8(reshape(ngan_matrix(:, n), im_m, im_m));
subplot(1,2,1)
imshow(im2), hold on
rectangle('Position',[1,1,100,100],...
    'LineWidth',3,'LineStyle','-', 'Edgecolor', 'r')
title('Original Image')
set(gca, 'Fontsize', 16)
subplot(1,2,2)
imshow(im1)
title('Background Subsection to Analyze')
set(gca, 'Fontsize', 16)

%% single generated person


%%

function curr_image = read_single_image(data, k)
    curr_image = imread(fullfile(data(k).folder, data(k).name));
end
```