Computational Notebook

Robin Sarah Chartrand
rchartra@uw.edu

March 7, 2024

Overview

This is a compilation of data analysis tools covered throughout this course. It goes through the four main topics covered in this course: transforms and time-frequency analysis, dimension reduction, an introduction to machine learning, and an introduction to deep learning. For each topic the theory, algorithms, and applications are explored and a few implementation examples are included.

1 Transforms and Time-Frequency Analysis

When we have a signal in the form of a time series it is often more useful to analyze the frequencies of the signal rather than the time series itself. Many transforms into the so-called "frequency domain" have been developed for this kind of analysis. The most well known is the Fourier transform, although there are many versions of the Fourier transform for different applications. The first is the Fourier Series. It transforms the signal onto a basis composed of cosine and sine functions over the interval $[0, 2\pi]$:

$$\hat{f}_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx$$

Arguably one of the most powerful properties of this family of transforms is they all have inverse transforms associated with them. This means that the signal can be changed in the frequency domain in some way, and then the original signal can be recovered with those changes now applied. For the Fourier series the inverse transform is defined as such:

$$f(x) = \sum_{k=0}^{\infty} \hat{f}_k e^{ikx}$$

This transformation can be extended to cover any interval including the entire real line $(-\infty, \infty)$. When this is done we get the Fourier Transform and the corresponding inverse Fourier Transform:

$$\hat{f}_k = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-ikx} dx$$

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(k) e^{ikx} dk$$

Most real world data however is discrete rather than continuous, as such the discrete Fourier transform and inverse transform was developed. It is an approximation of a the continuous Fourier transform:

$$\hat{f}(k_n) \approx \frac{1}{N} \sum_{n=0}^{N-1} f(x_n) e^{-i2\pi k_n n/N}$$

$$f(x_n) \approx \sum_{n=0}^{N-1} \hat{f}(k_n) e^{i2\pi k_n n/N}$$

As touched upon previously Fourier transform allows for the analysis of a signal in terms of its frequencies. Let us work through such an example. Consider the signal function:

$$f(x) = 2\sin(5x) + 8\sin(10x) + 5\sin(15x)$$

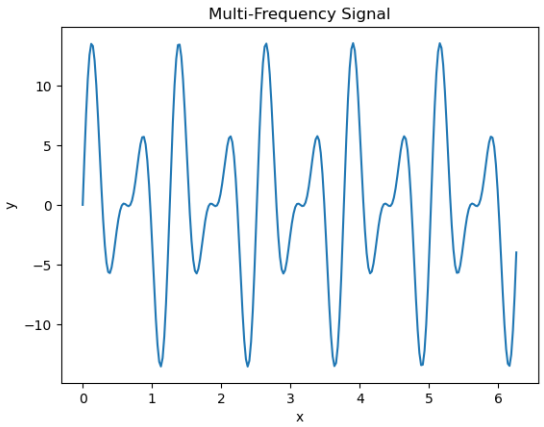

Figure 1: A signal time series composed of sin waves with different amplitudes and frequencies.

We know the signal is made up of functions with frequencies 5, 10 and 15. We also know the respective amplitudes are 2, 8 and 5. However say we were given this signal but did not know the original function. We can use the Fourier transform and frequency analysis to figure out the frequencies and amplitudes. The Python package Numpy [3] has various functions for performing fast Fourier transforms in various dimensions. For this problem we use the 1D function `np.ftt.fft()` to obtain the Fourier transform of the data. When planning to analyze the frequencies of the signal it is important to use the function `np.fft.fftshift()` on the Fourier transform to account for the order Numpy [3] outputs the frequencies in. If we plot the absolute value of the resulting frequencies we can see the dominant frequencies in the signal as well as their amplitude (Figure 2). We can see that three main frequencies are present in the signal at $|k| = 5, 10, 15$ with amplitudes of 2, 5, and 8. This matches the known frequencies of the original function.

The Fourier transform can be extended to multiple dimensions:

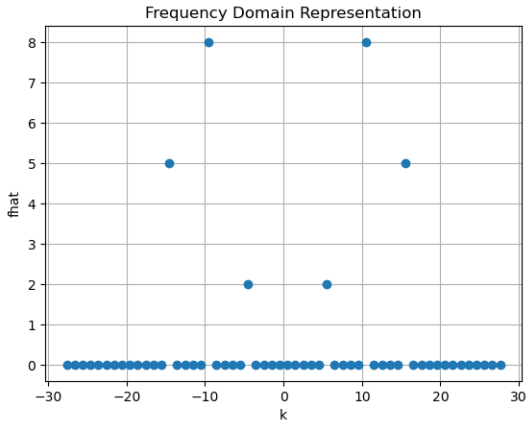

Figure 2: The signal from Figure 1 in the Frequency domain

$$\hat{f}(\overline{K}_n) \approx \frac{1}{N_x N_y \dots} \sum_{n_x=0}^{N_x-1} \sum_{n_y=0}^{N_y-1} \dots f(x_n, y_n, \dots) e^{-i2\pi k_{nx} n_x / N_x - i2\pi k_{ny} n_y / N_y \dots}$$

$\overline{K}_n$ is a tensor of the desired dimensions. This allows us to use the above techniques with multidimensional data as well. As an example, let us consider an example of denoising a color image (3 dimensions). As mentioned earlier, alterations can be made to the signal in the frequency domain that then manifest themselves when we recover the original signal using the inverse Fourier transform. This has a powerful application in denoising. When averaged over many samples, random noise in a signal tends to be of a lower frequency compared to the features of interest in the signal. As such, a filter that removes the lower frequencies can be applied to the signal in the frequency domain which will then remove the noise from the original signal once it is recovered. Let's say we have a noisy image as in the second image of Figure 3. We can take the Fourier transform of the data and apply a Gaussian filter centered over the notable high frequency point in the middle of the grid (Figure 4). When we then use the inverse Fourier transform to go back to the signal domain, we can see that while the denoising is not perfect we have recovered the vibrancy of the image and smoothed over the noise somewhat (Figure 5).

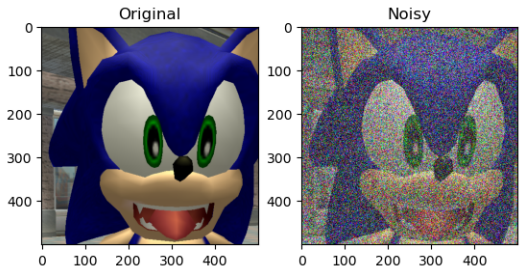

Figure 3: The original image and the same image but with random normal noise added.

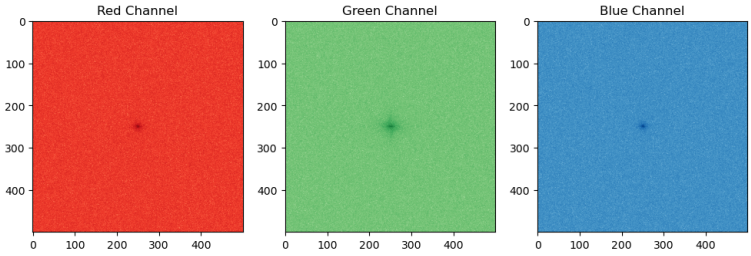

Figure 4: The Fourier transform of each of the three color channels in an RGB image. The colors are for identification purposes only and don't represent anything about the data itself.

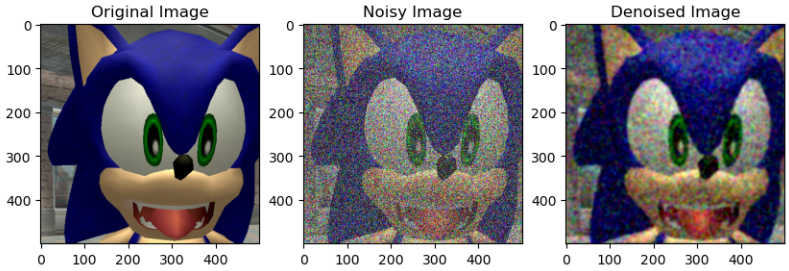

Figure 5: Comparison of the original image, the noisy image, and the denoised image to see the effectiveness of the denoising process.

2 Dimension Reduction

When attempting to analyze complex systems it is common to end up with high dimensional data with a large number of variables. While this seems to capture more information, it poses many problems when it comes to the analysis of such data. We can only visualize data up to three dimensions, and many algorithms run into "the curse of dimensionality" and don't perform as well with a large number of dimensions. There is therefore motivation to find a way to reduce the dimensions of a dataset while still capturing the majority of the information held by the variables. The solution to this problem comes from singular value decomposition (SVD). SVD decomposes a matrix A into three unitary matrices:

$$A = U\Sigma V^T$$

If we view A as an operator on a vector, U represents the change of basis into this new space, Σ is a diagonal matrix whose diagonal values, known as singular values σ, scale the vector, and V^T rotates the vector. The columns of U are special because when the data is properly centered they are the eigenvectors of the covariance matrix of a Gaussian distribution centered at zero. As such they are orthogonal vectors which represent the directions of greatest variance in the data in each dimension. These vectors are known as principal components, modes, or principal axes. The variance captured by each principal component can be calculated using the singular values by calculating the cumulative "energy" E_j of each additional mode j added to a representation:

$$E_j = \frac{\sigma_j^2}{E_{tot}}$$

$$E_{j,cum} = \Sigma_{i=1}^{j} \frac{\sigma_j^2}{E_{tot}}$$

This allows us to get a sense of how much information each additional principal component adds to a representation of the data. Often a relatively small number of modes will still capture the majority of the data's variance, allowing us to then toss out the majority of the modes. We are then left with a lower dimensional representation of the data that still keeps most of the information from the original data. We can then use this approximation for any further data processing (one often reduces the dimensions of the data before doing any machine learning) or we can reconstruct the original data from this reduced PCA mode representation to get a compressed version of the original data.

Let's illustrate these principles with an example using a dataset of 809 images of different Pokemon [11]. This dataset was selected as a challenge since despite being all from the same game, Pokemon all look wildly different from each other, so it will be interesting to see if PCA can find enough in common between them all to be able to compress this dataset. Each RGBA image is 120x120 pixels, and has four color channels so the array of images is reshaped into an array of size (809, 14400, 4). PCA is then performed on each color channel individually [10]. We use the Scikit-learn [9] PCA() object on each array of size (809, 14400) to get the singular values σ and the principal components (the columns of U). Each principal component is a vector of length 14400 so we can recombine the four channels and reshape each principal component into an image. This allows us to visualize the principal components themselves (Figure 6). Each component doesn't look like a specific pokemon anymore, but rather a colorful, vaguely pokemon shaped blob. This makes sense since the images represent the directions of greatest variance between each pokemon, so they shouldn't look like any one pokemon in particular. We can also use the singular values to calculate the cumulative energy captured with each additional PCA Mode (Figure 6). The cumulative energy is calculated for each channel separately, but all channels have about 95% of their variance captured by just 500 modes. We can use the first 500 components to reconstruct the original images and we see that this indeed gives up a pretty good approximation of the original images (Figure 7).

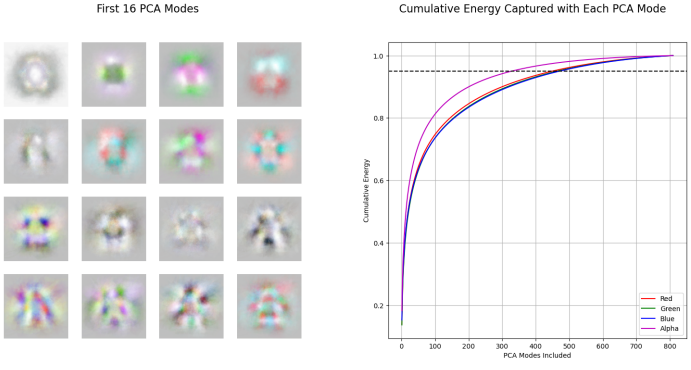

Figure 6: a.) The first 16 principal components shaped into images. b.) The cumulative energy captured by each additional principal component for each color channel.

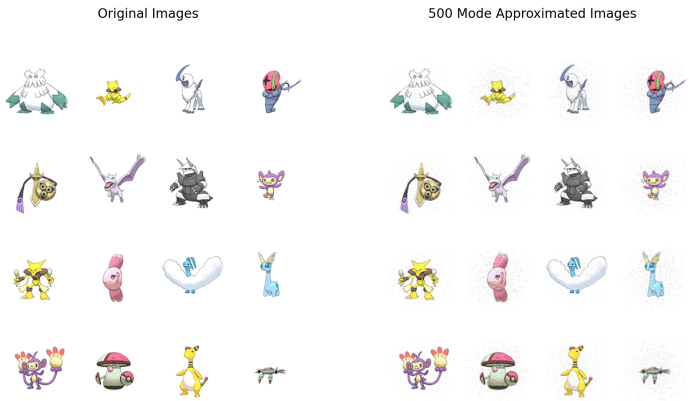

Figure 7: The first 16 original images compared with the first 16 images approximated using PCA.

3 Classical Machine Learning Methods

Machine learning is a term corresponding to a variety of algorithms that all "learn" how to perform a certain task. Machine learning methods are divided into two categories: supervised and unsupervised learning. Supervised machine learning methods are still quite diverse but the problems they solve are all set up in a similar way. The goal is to have the machine learning model teach itself with data on which you have already performed your classification, regression, or etc. task on, and then you can use your new model to perform that task on new data. Typically one has a dataset containing various inputs, variables, features, etc. that has a corresponding set of outputs or labels associated with it. One then divides this dataset into two, sometimes three subsets. The first dataset is called the training set, with the features labeled as X_{train} and the labels labeled as y_{train}. This is the dataset you use to train your model to perform your task. The second subset that is sometimes used is the validation set, which is also divided into X_{val} and y_{val}. This dataset is used to test the model during the training process to see how well the model is performing throughout different stages of the training process. The final subset is the testing set, which is once again divided into X_{test} and y_{test}. This dataset is used as a final test of the model's performance once it has been finalized. This rigorous oversight of the model is what gives it the name "supervised learning", and it is done to help combat a major struggle of machine learning: overfitting. One can highly optimize a model to be very good at performing the desired task on the dataset used to train it, however this doesn't always translate to being able to perform the task well on new samples. As such, constantly testing the model on data not used in the training process ensures that the model remains applicable to new data. Supervised learning is commonly used for tasks such as recognition, classification, regression, forecasting, and approximating.

Unsupervised learning is done in the case where one has a dataset of various features but no set of labels. Instead of learning how to assign known labels accurately, unsupervised learning is used to understand what useful information the dataset contains and come up with a corresponding set of labels for the data. This subset of models is also quite diverse, and includes methods such as Fourier transforms, dimension reduction, clustering, interpolation, and prediction algorithms. As we have already mentioned some of these models, we

will focus more on supervised learning methods.

Linear regression is one of the simplest and most fundamental supervised learning methods. Its goal is to find the optimal coefficients $\vec{\beta}$ that minimize the error of the model:

$$\vec{y} = \beta_0 + \sum_{j=1}^{d} \beta_j \psi(\vec{x})_j$$

While it is called linear regression, the model does not necessarily need to be linear. $\psi(\vec{x})_j$ can be any set of basis functions, it is the parameter model that is linear. The optimal coefficients are found by using least squares regression:

$$\beta^* = \arg\min_{\beta} \frac{1}{2\sigma^2} ||A\vec{\beta} - \vec{y}||^2$$

$$\frac{\partial}{\partial \beta}(\arg\min_{\beta} \frac{1}{2\sigma^2} ||A\vec{\beta} - \vec{y}||^2) = 0$$

$$\frac{1}{\sigma^2} A^T (A\vec{\beta} - \vec{y}) = 0$$

$$\beta^* = (A^T A)^{-1} A\vec{y}$$

This can run into issues however with large datasets as $A^T A$ has a high chance of being singular. To resolve this, we can use a method known as Ridge regression. We can add a regularization term to relax the singularities and ensure we can always take the inverse:

$$\vec{\beta}^* = \arg\min_{\beta} \frac{1}{2\sigma^2} ||A\vec{\beta} - \vec{y}||^2 + \frac{\lambda}{2} ||\vec{\beta}||_2^2$$

$$\vec{\beta}^* = (A^T A + \sigma^2 \lambda I)^{-1} A^T \vec{y}$$

The choice of λ has a large impact on the success of the model and it will ultimately depend on the data itself. However as a general rule it should always be kept as small as possible or else the model will be over regularized and unable to find optimal coefficients. Another supervised learning method is linear discriminant analysis. It doubles as both a method of performing dimensionality reduction and a classification method. It reduces the dimensions of the data and then maximizes the distance between means of classes and minimizes variance within classes, i.e. the ratio of between-class variance and the within-class variance [7]. It then identifies the linear combination of features that best divides the classes [4]. Another method known as K-means uses PCA to classify samples. It uses k PCA modes as a basis for a k-Mode PCA space and finds the centroids (k-dimensional mean) of each class in that space. It then classifies new samples by projecting them onto this k-Mode PCA space and determining which centroid it is closest to. A similar clustering method, KNN, compares the location of new data points in feature space to the training points. It then assigns the new point to the class the majority of its k nearest neighbors belong to. Its accuracy and efficiency are affected by the choice of k whose optimal value is highly data-dependent. For the majority vote method to work however k is always chosen to be odd [9].

These models all have a multitude of parameters that need to be adjusted as part of the training process. This process is part of hyper-parameter tuning, where you adjust the parameters of your model and see the results on your model's accuracy. An important part

of this is having good accuracy measures. A common measure for these classical models is the mean square error:

$$MSE(f, \vec{y}) = \frac{1}{N} \sum_{n=0}^{N-1} |f(\vec{x}_n) - \vec{y}_n|^2$$

Regardless of the parameters chosen, there will often be some random variation in model errors/accuracies with each model run. This can make it difficult to rigorously compare multiple models since some of the variation will be due to chance not an actual increase in performance. To account for this a common method to use is cross-validation. The method involves shuffling the samples and then dividing the training dataset and labels into some number of subsets. The model is then trained on each subset and errors/accuracies are calculated for each subset. The average error/accuracy as well as the standard deviation can then be calculated which allows for more accurate comparisons between models.

Let's use these methods for a classification example. I found a dataset of 757 images pulled from the Animal Crossing subreddit and 840 images from the Doom subreddit [6]. The images are mostly memes or screenshots from the game posted by users (Figure 15). The images are all different shapes and sizes so for simplicity they are all reshaped to a 300x300 RGB image using the Python PIL library [13]. The images are then flattened and compiled into one array of size 1597x270000. A corresponding vector of labels is made as well, labeling the Animal Crossing images with a zero and the Doom images with a one. The data is then randomly shuffled, split into training and testing datasets, and then scaled using the Sklearn StandardScaler() [9]. The data has a very large number of dimensions, so using PCA to reduce the dimensions is recommended. When PCA is performed we can see that we can capture 95% of the data's variance with just 578 modes. As such the original training and testing data were projected onto 578-mode PCA space before being used by the models. Ridge classification, Linear Discriminant Analysis, and K-Nearest Neighbors were all tested on this dataset. Each model was implemented using its corresponding method in the Sklearn Python library [9]. The models were each trained on the training dataset, then tested on the training and testing datasets, and then cross validated on five subsets of the training dataset. The results are illustrated in figure 8. We can see that KNN performed better overall in terms of the cross validated accuracy though with a wider range of variation than the other two methods which both had roughly the same cross validated accuracy of around 61%. KNN also had the least overfitting of the three models. Still, overall, these classical models were alright but not that good at classifying between the two video games as they all had nearly the same testing accuracy of around 68%. This could be for an abundance of reasons. This dataset is relatively small compared to more common machine learning benchmark datasets like MNIST. These classical models also aren't very powerful when it comes to complicated datasets like this one that have a large number of dimensions. Hopefully, with some more powerful machine learning methods we can get some better results.

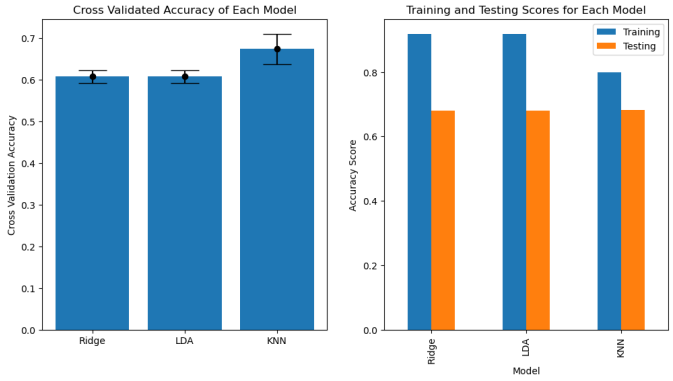

Figure 8: a.) The cross validated accuracies of each of the three models on five subsets of the training set. b.) The accuracy of each model on both the entire training set and the testing set.

4 Neural Networks and Deep Learning

For complex datasets or difficult computational tasks the aforementioned classical machine learning methods often aren't enough. For computational power we must turn to neural networks. Neural networks are structures made up of interconnected layers of computational units known as neurons. They perform classification on data and then optimize their own performance to increase their accuracy over many successive cycles known as epochs. Each neuron first takes in input from the neurons in the previous layer and then multiplies that input by a set of weights and adds a bias term. It then applies an activation function to the input data. There are many options for this function, but they all have a similar step like shape beginning at one value and transitioning to another at a certain point with varying degrees of smoothness. Some examples of such functions are shown in Figure 9. If the combined input is above a certain threshold, the neuron activates and passes the data on to the next layer. If not, then no data is passed forward. Neurons are grouped together in layers, which are combined to form a stack of an input layer, some number of hidden layers, and an output layer. Deep neural networks are networks that have a large number of hidden layers. While this creates a stronger model, it also creates a number of new issues that must be addressed throughout the training process.

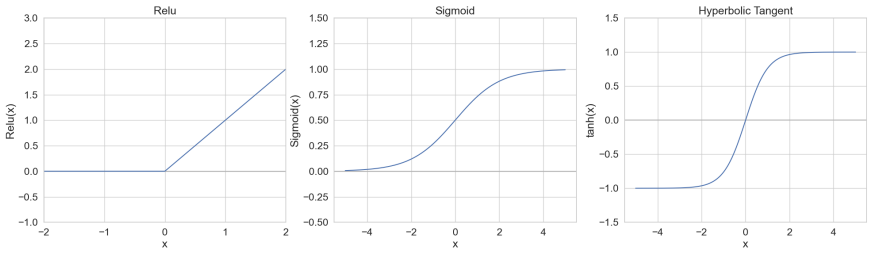

Figure 9: Three commonly used activation functions for neurons: Relu, sigmoid, and hyperbolic tangent.

To train the model, the training features are propagated through the neural network until they reach the output layer. Loss, or the model's current accuracy, is calculated via a loss function throughout the training process. There are many ways to calculate loss, though they all work by comparing the classification labels produced by the model to the known labels of the training set in some way. One such loss function that is commonly used is the cross entropy loss. It is geared for working with probabilities as it compares the distributions of the results rather than the values themselves and calculates the loss logarithmically:

$$L(y_{pred}, y_{true}) = -(y_{true}\log(y_{pred}) + (1 - y_{true})\log(1 - y_{pred}))$$

The loss function is optimized by an optimization function to find the weights and biases that will reduce the loss function in the next epoch. The pace at which optimization occurs can be controlled via a learning rate parameter α that should be tuned according to the specific dataset being used. The network is then updated with the new weights and biases through a process known as back propagation. There are many optimization functions available, however some commonly used ones are stochastic gradient descent (SGD), Adam, and root mean square propagation (RMSProp). SGD uses the gradient descent optimization method (shown below) and updates the weights and biases of the model each time the loss is calculated.

$$J = \frac{1}{n}\sum_{i=1}^{n} L(y_{pred,i}, y_{true,i})$$

$$\vec{\omega}_{k+1} = \alpha \nabla_{\vec{\omega}} J(\vec{\omega}_k; b)$$

$$b_{k+1} = b_k - \alpha \frac{\partial}{\partial b} J(\vec{\omega}; b_k)$$

Adam and RMSProp are more advanced versions of SGD that adjust the learning rate during the training process according to the calculated gradients to better minimize the loss [1]. There are many parameters to determine in the aforementioned structure and training process. The adjustment of these parameters as well as the implementation of additional performance improving methods is another example of the hyper parameter tuning process. Some common methods to employ are dropout regularization, initialization, and batch normalization. Dropout regularization works by randomly turning off neurons in a layer according to some probability. This can help with overfitting by preventing the network from relying on just a few neurons overly specialized to the training data and forcing the other neurons to also be a part of the classification process [12]. Initialization is the strategic setting of the initial weights for the network in such a way that improves the optimizer's chance of converging to the global minimum. Different initialization methods are known to work better for different activation functions. For example the Xavier normal initialization method is known to work better for hyperbolic tangent and sigmoid activation functions, and the Kaiming He method is known to work better for the ReLU activation function [2]. Finally, batch normalization is a technique that stabilizes networks and prevents the divergence of the gradients towards 0 or infinity. This is especially important for deep neural networks with a large number of layers as each new layer increases the odds of the gradient vanishing or exploding. It is done by normalizing the outputs of each layer to fix the means and variances of layer inputs which ultimately stabilizes the model [5].

To illustrate some of these principles let us return to the classification problem tackled previously. The goal is to train a classification algorithm to differentiate between Reddit

image posts about Animal Crossing and posts about Doom. We once again project the dataset onto the lower dimensional 578-mode PCA space, and then divide the dataset into training, validation, and testing subsets. We will then build a neural network model, train the model over a number of epochs validating at each step, compute the final model's accuracy, and then go back and tune the parameters of the model to increase the accuracy. This is all done using the Pytorch Python library [8] by defining a class with the aforementioned model structure. For this task, an input layer with 578 neurons, an adjustable number of hidden layers with an adjustable number of neurons, and then an output layer with 2 neurons were each defined in the `__init__()` function of the class. The forward function of the class was then defined which passes the model input through each layer and defines the activation function for the neurons. After some experimentation the Relu activation function was chosen for this task as it was clear it had the best results. Through more experimentation, it was found that three hidden layers of size 400, 200 and 200 respectively worked well for this model and dataset.

The class was then setup to be able to perform, or not perform, the following hyperparameter tuning methods: dropout regularization, various initialization options, and batch normalization. The model was also set up to use an adjustable learning rate and one of three optimization functions: SGD, RMSprop and Adam. Once the model was defined a new function was created to train the model over one-hundred epochs. The function propagated the training dataset through the model, calculated the loss, optimized the loss function, propagated the new weights back through the model, and then calculated the validation loss and accuracy for each epoch. It also calculated the model's accuracy on the test set at the end of the training process. Using this function all the aforementioned parameter options were tested. First the three different optimization functions were compared as well as four different learning rates. The model was run five times and then the average test accuracy and standard deviation were calculated (Figure 10).

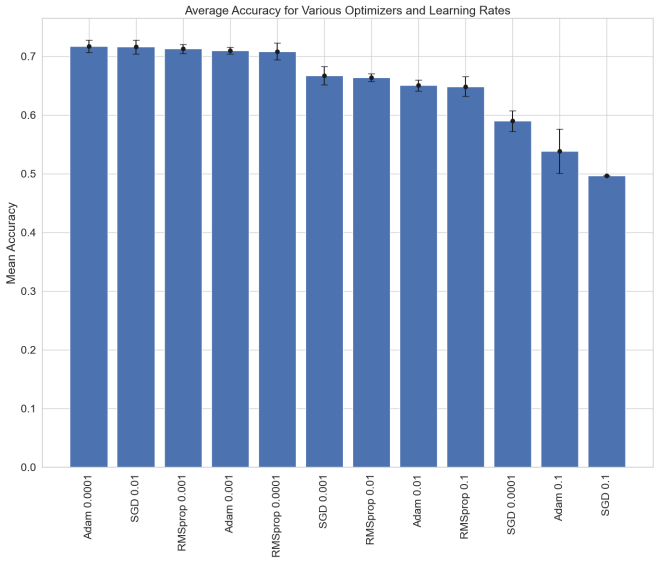

Figure 10: A comparison of the performance of the model using three different optimization functions and four different learning rates.

There were five configurations that stood out and they all performed roughly the same, hovering just above 70% accuracy. Ultimately the top result, the Adam function with a learning rate of 0.0001 was selected and used for the remainder of the tests. The effect of using dropout regularization and batch normalization was then tested in the same way (Figure 11). Neither method had any effect on the test accuracy. For the dropout method, the training and validation loss curves were also compared since dropout is known to fix over fitting (Figure 12). It's clear that the model exhibits loss curves characteristic of overfitting, and implementing the dropout method does indeed fix that. However, the loss curve for the drop out method is no longer smooth and the convergent loss value is much higher than ideal. The same pattern was found when the dropout probability parameter was adjusted. As such it was decided to not use batch normalization or dropout regularization in the final model with the understanding that the model was indeed overfitting to the training data. Finally, three initialization functions for the model's input were tested on the model: Random Normal, Xavier Normal, and Kaiming Uniform (Figure 13). It was found that no initialization function performed better than using no initialization function, and as such no initialization function was included in the final model.

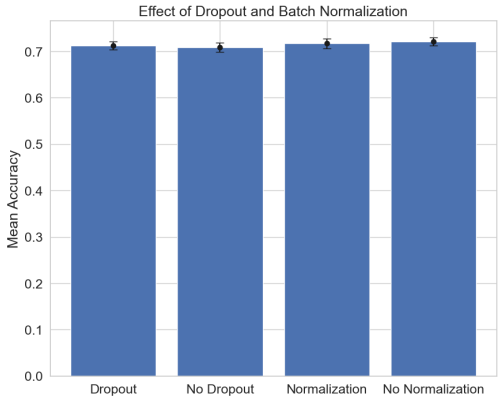

Figure 11: The model's accuracy on the test set using and not using dropout regularization and batch normalization.

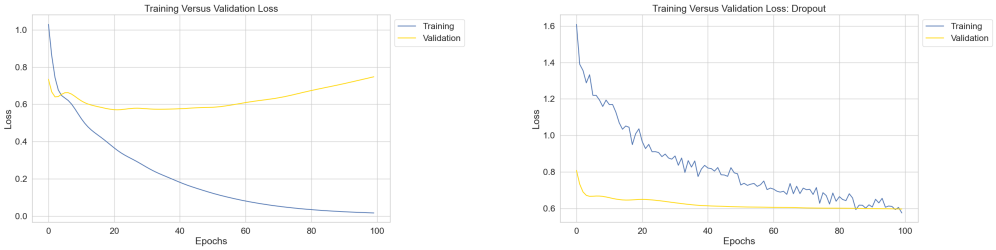

Figure 12: a.) The training and validation loss curves for a model run without dropout b.) The training and validation loss curves for a model run with dropout.

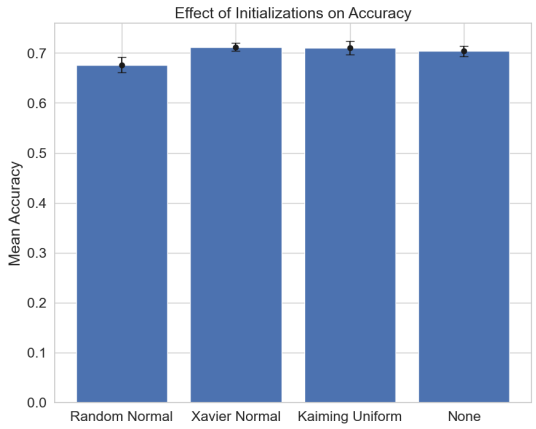

Figure 13: The results on the model's accuracy on the test set of three different initialization functions, as well as the results without an initialization function.

Ultimately, many of the hyperparameterization methods did not boost the performance of the model. The final model had an accuracy of around 71% and had the training loss curves and validation accuracy curves shown in Figure 14. The model performed slightly better than the classical methods used in section three which were around 3% less accurate. However it still did not perform as well as one would hope and the model could not be adjusted above it's base accuracy. It also exhibited an obvious trend of overfitting. All these issues likely stem from same issue which is the small sample size of the data set. Neural networks get their strength from finding patterns in the data it can use to differentiate between the classes. These patterns are hard to find when there aren't a lot of samples to work with. The classification problem presented is also quite difficult. While the origin of the content of the images is easily discernible to the human eye, many of the meme style images have a very similar look and format (Figure 15). As such using more powerful computational methods or larger neural networks likely would not increase the accuracy by much more.

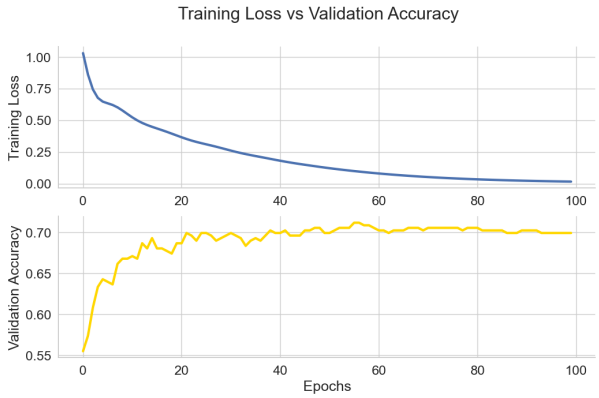

Figure 14: The training loss and validation accuracy curves of the final model.

Figure 15: Two very similar images from the Animal Crossing subreddit (left) and the Doom subreddit (right).

5 Conclusions

In an era of research where data is becoming increasingly larger and more abundant, data analysis techniques are critical to answer the questions we ask. There are many such techniques for a variety of purposes. Fourier Transforms and time frequency analysis allow for more informational perspectives of signals from a domain that is more suited to the periodic nature of such data. Dimensional analysis allows us to better understand the information captured by a dataset and provides the very useful technique of dimension reduction. Machine learning is a powerful computational tool for a wide range of applications such as regression, classification, prediction, and clustering, just to list a few. While such techniques have different purposes there are many similarities between them and many of them build off of each other. For example dimensional analysis is often a precursor to many algorithms in order to reduce dimensions of data, but there are also many algorithms who take advantage of the properties of PCA and the k-mode PCA space one can create with it. These are just a few of a multitude of insightful techniques that have been developed and much new research is still to come in the further development of these techniques.

6 Acknowledgements

The author would like to thank Professor Shlizerman for introduction to the topics in this paper and his direction on the implementation of these methods in real world examples. The author would also like to extend their gratitude to Juan Ramirez and Saba Heravi for their advice and discussion on these topics throughout the course. Thank you for a wonderful quarter!

References

[1] J. Brownlee. Gentle introduction to the adam optimization algorithm for deep learning, Jan 2021.

[2] J. Brownlee. Weight initialization for deep learning neural networks, Jan 2021.

[3] C. R. Harris, K. J. Millman, and et. al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[4] IBM. What is linear discriminant analysis?, Nov 2023.

[5] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[6] Larxel. Doom or animal crossing? (image classification), 2020.

[7] N. Mohanty, A. L.-S. John, R. Manmatha, and T. Rath. Chapter 10 - shape-based image classification and retrieval. In C. Rao and V. Govindaraju, editors, *Handbook of Statistics*, volume 31 of *Handbook of Statistics*, pages 249–267. Elsevier, 2013.

[8] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[9] F. Pedregosa, G. Varoquaux, and et. al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] R. Raj. Image compression using principal component analysis (pca).

[11] V. Subbiah. Pokemon image dataset, 2024.

[12] A. Tam. Using dropout regularization in pytorch models, April 2023.

[13] P. Umesh. Image processing in python. *CSI Communications*, 23, 2012.