INSTANT EVALUATION OF LIVE CAR-T CELLS USING RAPID QUANTITATIVE PHASE IMAGING AND MACHINE LEARNING

by

RACHEL NICOLE CARLYLE

(Under the Direction of Peter Kner)

ABSTRACT

The increasing prominence of cell therapy in oncological treatment necessitates innovative, non-destructive methods for efficient cell manufacturing. Current evaluation techniques are often destructive or incompatible with cell therapies, a limitation addressed by Quantitative Phase Imaging methods such as Differential Phase Contrast (DPC). DPC enables high-resolution, label-free imaging of live cells. In combination with machine learning algorithms, this technology permits real-time, non-invasive evaluation of therapeutic cells, including Mesenchymal Stem Cells (MSCs) and Chimeric Antigen Receptor T-cells (CAR-T). A study was conducted involving anti GD2 and mCherry CAR-T cells, employing DPC and Fourier Ptychography for imaging over three days. Images were reconstructed using Python, and cellular segmentation was conducted via CellProfiler. The machine learning techniques, KNN, Random Forest, and Autoencoder, were then used for classification. The autoencoder achieved classification accuracies of up to 93% on aggregate data, thereby validating this integrated approach for quality control in therapeutic cell manufacturing.

INDEX WORDS: Fourier Ptychography, Differential Phase Contrast, Microscopy, Quantitative Phase Imaging, Machine Learning, CAR-T Cells

INSTANT EVALUATION OF LIVE CAR-T CELLS USING RAPID QUANTITATIVE PHASE IMAGING AND MACHINE LEARNING

By

RACHEL NICOLE CARLYLE

B.S., The University of Georgia, 2022

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2023

INSTANT EVALUATION OF LIVE CAR-T CELLS USING RAPID QUANTITATIVE PHASE IMAGING AND MACHINE LEARNING

By

RACHEL NICOLE CARLYLE

Major Professor:	Peter Kner
Committee:	Luke Mortensen
	Lohitash Karumbaiah

Electronic Version Approved:

Ron Walcott
Vice Provost for Graduate Education and Dean of the Graduate School
The University of Georgia
December 2023

DEDICATION

I’d like to dedicate this thesis to Alan Carlyle, for all love of science and engineering he fostered in me growing up, and his steadfast support in all my pursuits.

ACKNOWLEDGEMENTS

In 2018, Luke Mortensen answered an email and gave an undergraduate a chance to do research in his lab. I'd like to thank him and Kayvan Tehrani for starting my path to graduate school with undergraduate research, Peter Kner, for his unwavering support and encouragement in my academic pursuits, my family for all the support they've given me, and all the kind friends who have delivered food and coffee while I've been up later than I'd like to admit writing this.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

AN OVERVIEW OF BIOMANUFACTURING, CELL-BASED THERAPIES, CAR-T CELLS, QUANTITATIVE PHASE IMAGING, AND MACHINE LEARNING

1.1 Importance of Biomanufacturing

Regenerative bioscience is an exponentially growing field that aims to understand the complex mechanisms behind how the body repairs itself and to utilize this knowledge to treat incurable diseases and debilitating injuries [1, 2]. Living cells are used to manufacture medical products in a controlled and reproducible manner, otherwise known as biomanufacturing [3]. As scientists look to commercialize their research and bring biotherapeutics to market, scale-up and scale-out manufacturing techniques become critical. The utilization of closed-system bioreactors, high-throughput screening, and automation has elevated the scalability and precision of bioproduct production and increased modularity [4, 5]. For instance, disposable bioreactors and single-use systems have helped minimize cross-contamination risks and reduced the time required for system setup, accelerating production cycles [6]. Advancements in biomanufacturing technologies enable the mass production of targeted therapies and directly impact the availability and cost-effectiveness of cancer treatments that utilize therapeutic cells, such as Chimeric Antigen Receptor T (CAR-T) cells.

1.2 CAR-T Cells

T cells, crucial to the immune response, are extracted from the patient's blood through leukapheresis, then genetically modified via a viral vector to express the Chimeric Antigen Receptor (CAR) on the cell's surface [7, 11]. The receptor is a synthetic chimeric protein

containing multiple engineered elements, such as an extracellular antigen binding domain, a hinge region, a transmembrane domain, and an intracellular signaling domain [7, 11, 12, 13]. This results in a T cell engineered to target a specific antigen, allowing for a highly personalized approach to cancer treatment, even in patients with refractory cancers.

This cell, which recognizes a specific antigen on the surface of cancer cells, is then reinfused into the patient to elicit a cytokine response [13]. This CAR allows the modified T cells, or CAR-T cells, to target cancer cells as part of the patient's immune response by recognizing specific antigens expressed on the surface of cancer cells, facilitating their destruction. In research studies, these cells co-express mCherry, a fluorescent protein, to report successful transduction, acting as a reporter gene. Current CAR-T cell therapies offer a novel and potentially curative treatment for patients with refractory or relapsed malignancies [14].

Specifically, the CAR-T cells used in this research are modified with the GD2 receptor. Mutated glioma cells, such as those in Glioblastoma, a malignant brain tumor, overexpress disialoganglioside 2, or GD2, on the cell's surface [9]. Knowing the specific antigen a cancer cell expresses, the CAR-T cells can be modified to target this specific antigen. Cells that express GD2 are considered anti GD2 and can be a suitable candidate for CAR-T cell therapy [7, 9]. CAR-T cell therapy has proven especially useful in treating hematological malignancies like leukemia and lymphoma, specifically particular types of B cell lymphoma and treatment-resistant acute lymphoblastic leukemia [7, 8].

Biomanufacturing is pertinent in cancer treatment, where the need for individualized, high-quality, and scalable solutions is pressing. Specifically, in the case of CAR-T cell therapies used in cancer treatments, which are popularly used for treating types of blood cancers such as diffuse large B-cell lymphoma (DLBCL), follicular lymphoma, mantle cell lymphoma, multiple myeloma,

and B-cell acute lymphoblastic leukemia in pediatric and adult patients [7,8, 9], there are strict GMP (Good Manufacturing Practices) that must be adhered to due to the complexities involved with cell isolation, gene editing, and cell culturing [10]. Many factors involved in cell isolation, transduction, and expansion must be controlled and monitored to ensure quality and safety control before they are reinfused into the patient [11, 12].

1.3 Evaluating Therapeutic Cells

Demonstrating safety and efficacy of therapeutic cells is critical to achieve FDA approval. There are several problems that need to be addressed to ensure that these cells are safe and effective as scaling up in the manufacturing process becomes important and introduces more variables. Some of these problems include functional heterogeneity, or "differences in therapeutic function of cells from different donors, tissue sources, and manufactured under myriad conditions", lack of critical quality attributes, and the effects of changes when manufacturing is scaled up, such as different culture media, culture vessels, cryoprotectants, and isolation techniques [40]. Cells can vary in quality within the same donor as well, further increasing the need for reliable markers of quality. The lack of defined CQA's makes it difficult to predict how these therapies will perform and contributes to the difficulty of scale up manufacturing and gaining FDA approval [40].

Some of the common criteria currently evaluated to ensure the efficacy and safety of these cells include phenotypic markers, viability markers, potency assays, purity and contamination assessment, genomic integrity, and functional assays [15, 16, 17, 18]. Phenotypic markers can be evaluated using flow cytometry, and protein expression on the cell surface can be examined to confirm that the CAR was successfully integrated [17, 19, 20, 21]. Flow Cytometry has the advantage of being a high throughput method and can assess other parameters such as size and examine multiple fluorescence markers [17, 19, 20, 21]. Viability markers as byproducts of

metabolic activity are evaluated, gauging the cells' overall health and functional potential [15, 16]. Potency assays can include in vitro cytotoxicity tests and in vivo models to test capabilities such as longevity and immunological memory potential [15, 16, 18]. Purity and contamination are assessed using PCR and mass spectrometry, testing for anything that could compromise these cells' safety [21]. Genome sequencing detects any unwanted mutation or chromosomal defects that would affect long-term safety [21]. Some of these methods, however, are invasive, destructive, costly, and time consuming.

1.4 Other Imaging Techniques and Quantitative Phase Imaging

As part of this process, various imaging techniques have been applied, namely fluorescence microscopy techniques such as high content imaging, two-photon and multiphoton microscopy, and quantitative phase imaging (QPI) methods like Differential Phase Contrast (DPC) and Fourier Ptychography Microscopy (FPM).

Widefield Fluorescence microscopy can evaluate multiple cell markers, but can also photo bleach and damage the samples [22]. High content imaging is a valuable imaging technique that can take thousands of single cell images and quantify single cell morphological features utilizing high contrast fluorescent cell labeling techniques [39]. While fluorescence microscopy does provide high resolution images, it requires fixing and staining the cells, therefore only providing a snapshot of morphological data from the cell state at the point in which it was fixed [40].

Two-photon microscopy is costly, and while it can track CAR-T cells within tissues and cell behaviors in real time for in vivo studies and provide single cell resolution, few fluorophores are suitable for multiphoton excitation [22]. There are some research efforts to utilize it as a non-destructive, in process imaging technique with two photon autofluorescence lifetime imaging to evaluate and classify T- cell activation [41, 42, 43]. This method images endogenous metabolic

co-enzymes NADPH and FAD, with NADPH representing the combined fluorescence signal [41, 43]. The imaging is done while the fluorophore is in the excited state before returning to ground state, allowing for information on the protein binding of NADPH and FAD. Activated T- cells have increased metabolic activity, and this imaging method provides quantitative features that allow for classification of activated T-cells in a non-destructive manner [41, 43]. This technique was also performed with time domain single photon excited autofluorescence lifetime imaging and was able to achieve a higher acquisition speed than multi-photon fluorescence lifetime imaging [44].

Conversely, quantitative phase imaging (QPI) techniques allow non-invasive analysis of in vitro biological samples [23, 40]. These methods, such as DPC and FPM, are label-free and non-destructive [23, 40]. FPM combines several low-resolution images with different illumination angles to reconstruct a high-resolution, complex image that includes both amplitude and phase [23, 24]. While it can achieve high spatial resolution and wide field of view, it is computationally costly, needing complex algorithms and more computational power to reconstruct the images [23].

DPC imaging is done via an asymmetric illumination scheme that converts phase gradients into intensity variations in the image [25, 26]. It quantifies the phase shift of light that passes through the sample [25, 26]. A quantitative phase profile of the sample can be reconstructed. This method is rapid compared to FPM and can provide real-time images of live samples [27]. FPM and DPC can be performed with an LED array, which can also provide brightfield and darkfield images, all on a single microscope [27].

DPC does not require staining and preserves the cells in their native state and can help evaluate CAR-T cells' overall health and viability while being non-destructive. It allows for detailed observation of cell characteristics and opens the potential for longitudinal studies and in

process monitoring, making it ideal for imaging CAR-T cells and possibly greater feedback control during their manufacturing [40]. DPC and FPM imaging techniques can be employed on a microscope with an LED array, making it a cost-effective method that allows for brightfield, darkfield, FPM, and DPC imaging on one microscope [27].

These methods can also be modified to be high throughput and integrated with machine learning algorithms to evaluate and interpret data from the images [19]. Both of these methods however, since they are non-invasive and label-free, lack specific labeling of features in cells that fluorescence imaging techniques provide. Instead, phase imaging provides information on absorption and morphology.

With many imaging techniques having the capability of evaluating cells metabolic activity and assessing morphology to determine functionality, morphology is a promising CQA, especially when coupled with non-invasive imaging methods [40]. Non-invasive imaging methods also provide a more flexible, cost and time effective method of in line evaluation compared to previously discussed assays.

Research on the cell manufacturing processes and optimization of these CAR-T cells is crucial for advancing personalized medicine. It can enhance their efficacy, scalability, and safety and improve quality control. Using non-invasive imaging systems to monitor the CAR-T cells and evaluate quantitative data to determine successful transfection of the CAR gene, viability, and potency without damaging the limited supply of CAR-T cells would significantly improve the manufacturing process of CAR-T cells. Current methods of assessing CAR-T cells are destructive, typically involving staining the cells. A high throughput, high accuracy method of assessing these cells is essential for improvement.

1.5 Machine Learning

Many methods of evaluating CAR-T cells rely on time and training in intensive labeling and staining techniques with specific biomarkers that affect cellular viability and cell signaling [19, 36]. Current label free assays evaluate a single feature and are low throughput [19]. A need for a high throughput, non-invasive, label free way of assessing and differentiating these cells based on multiple features is needed. Many machine learning models are successful using morphology to classify cells [35, 37, 38, 40, 41, 42] which quantitative phase imaging can provide [23, 34, 35, 36]. Using machine learning in conjunction with high throughput, quantitative phase imaging methods could provide a label-free, non-invasive modular imaging solution that allows for the evaluation of multiple critical quality attributes, ensuring the safety and efficacy of therapeutic cells [40, 42]. When analyzing data and making decisions and predictions, machine learning, a subfield of artificial intelligence, is a popular choice for examining complex, high-dimensional data [28, 32]. There are many different learning algorithms, such as supervised and unsupervised. Supervised learning methods, such as decision trees, and K-Nearest Neighbor (KNN) use labeled data to train the model, and adjustments are made based on differences between the predictions and actual values [28, 31].

A popular decision tree method is random forest [29]. For classification tasks, the output of the random forest is the class selected by most trees. The training data is fed to train various decision trees, and features from the data will be selected randomly during the splitting of nodes [29]. Every decision tree consists of decision nodes, leaf nodes, and a root node, with the leaf node of each tree being the final output or decision. The output, or final decision, is based on the output of multiple decision trees [29]. A 2021 study used random forest for cell classification, aiming to differentiate healthy from apoptotic cells from a human colorectal cancer cell line using data from

flow cytometry [35]. After staining and analysis, using a random forest model, this study was able to attain a good predictive performance. It predicted live cells with 93.2% precision and a 91.1% recall, and 91.4% and 89.1% for late apoptotic cells, respectively [35]. Another 2021 study using two-photon autofluorescence lifetime imaging achieved 97-99% accuracy in classifying T cells according to activation state [41].

KNN operates under the assumption that similar data points tend to have similar values. It stores an entire training data set to reference rather than undergoing a training process and classifies new, unlabeled data based on proximity to the defined classes in the training data set [31]. A 2023 study used a KNN algorithm to detect and classify poikilocytosis affected red blood cells of various shapes [37]. Specifically, this study utilized a hyper parameter optimizable KNN algorithm. When tested on two different data sets, they were able to achieve accuracy, precision, and recall scores above 97% [37]. Another study in 2021 used KNN to classify specific subtypes of myeloid leukemia. This method was less successful due to the similar morphology of some subtypes, resulting in an accuracy of 80.55%, and recall and precision scores of 44.15% and 42.59% respectively [38].

Unsupervised learning is popular for determining the structure or distribution of data without providing labeled training data, such as k- means clustering [30]. k- means clustering searches for a predetermined number of clusters in an unlabeled data set, aiming to organize the data space so that data points in the same class or cluster are as similar as possible [29]. The same 2021 study that used random forest also employed k- means clustering. When performed on standardized data for predicting live cells, they achieved a precision of 50.3%, 62.8% for recall, and 58.5% for accuracy [35]. Overall, this study compared multiple machine learning models to develop a more robust, multilayer perceptron model for predicting cell states [35].

Neural networks are a subset of machine learning consisting of layers or nodes connected through a series of weighted connections [28, 32, 33]. Autoencoder, a type of artificial neural network, is a fundamental building block of many machine and deep learning models for unsupervised learning tasks involving dimensionality reduction and feature learning without labeled training data but can also be supervised or semi-supervised depending on the specific model [32]. Overall, an autoencoder learns a compressed representation of the input data and then reconstructs the original data from the compressed representation [28, 32, 33]. The autoencoder consists of two main components: the encoder and the decoder [28, 32, 33].

The encoder is the neural network that transforms the input data into a lower dimensional representation, or the latent space, by learning features containing the most crucial input data information to reconstruct it [28, 32, 33]. The latent space can contain valuable insights about the structure or features of the data. The decoder is a second neural network that takes this compressed representation and reconstructs the original data [28, 32, 33]. It learns to map the lower dimensional representation to the high dimensional data space. It is trained by minimizing the differences between the input and the reconstructed output data, typically using a loss function such as mean squared error or binary cross entropy [28, 32, 33].

Prior studies have examined the use of label-free cell classification using flow cytometry and deep learning, with some models having over 85% mean accuracy [19] and one weakly supervised model having 100% classification accuracy [20]; however, many models using flow cytometry utilize manually labeled, single-cell images. Another study used autofluorescence intensity images and convolutional neural networks to classify T- cell activity [45]. Their specific method of fine tuning a pre-trained convolutional neural network resulted in a 98.83% accuracy, 99.14% precision, and 95.85% recall [45]. With DPC imaging, sample images of thousands of

labeled cells can train a semi-supervised deep learning model to classify CAR T cells based on morphological and quantitative phase and intensity data.

1.6 Purpose of Study

While there are many valuable methods of evaluating CAR-T cells for therapeutic use, there is still an ongoing need for an easy-to-use, noninvasive, nondestructive, and high-throughput method of assessing them. I will use two quantitative phase imaging methods, Differential Phase Contrast Imaging and Fourier Ptychography Microscopy, to evaluate human T cells for successful transfection of the GD2 CAR. Once these cells have been imaged, I will segment them using CellProfiler, extracting 155 features on morphology, proximity to nearest neighbors, and intensity. These features will be imported into a semi-supervised autoencoder to determine critical features for classifying these cells as either GD2 positive or GD2 negative. This method opens the potential for low-cost, high-resolution imaging of large quantities of CAR-T cells without the need for removal from a bioreactor or incubator, decreasing the amount of product loss in the cell manufacturing process.

1.7 Outline of Thesis

Chapter 1 will provide an overview of the importance of biomanufacturing in cell manufacturing for cell-based therapies, important markers for evaluating therapeutic cells, and existing methods of evaluating these cells, including flow cytometry, mass spec, MRI, and Fluorescence imaging methods. Then, there will be a brief overview of Quantitative Differential phase contrast microscopy, Fourier Ptychography microscopy, and machine learning methods.

Chapter 2 presents the details of FPM and DPC imaging and a description of the fundamental mathematics behind these existing reconstruction methods. An overview of the theory behind Autoencoders and network structure will also be given.

Chapter 3 will cover the methods for cell culturing done by a partnering lab, imaging methods, the use of CellProfiler, a semi-supervised autoencoder, and a description of the microscope setup.

Chapter 4 presents the experiment results, reviewing cell segmentation and evaluating the classification results using an autoencoder.

Chapter 5 summarizes the results, presents conclusions, and provides recommendations for the continuation of the work.

CHAPTER 2

DIFFERENTIAL PHASE CONTRAST AND FOURIER PTYCHOGRAPHY

2.1 The Widefield OTF

In widefield microscopy, which relies on variations in the absorption of light by the sample, imaging biological samples can be challenging due to the low absorption, limited contrast, and the presence of out-of-focus light [25, 27]. Thin samples often result in phase shifts rather than absorbing and scattering light. MSCs or CAR-T cells, which can be considered thin samples, have minimal light absorption and scattering, making them difficult to image with widefield microscopy [25, 27]. The response of the microscope to a sample can be mathematically described by the optical transfer function (OTF).

The (OTF), is a complex valued function which serves as a mathematical representation of the response of an imaging function as a function of spatial frequency. It is the Fourier transform of the point spread function (PSF), and it describes how modulation and phase change as a function of spatial frequency. It is generally broken down into two components, the modulation transfer function (MTF) and the phase transfer function (PTF), representing how the amplitude and phase of the spatial frequencies are altered.

To elucidate this point further, consider the following equations that describe imaging in real and Fourier space. In real space, the image $I(r)$ is formed by the convolution of the object $o(r)$ with the point spread function $h(r)$:

$$I(r) = o(r) \otimes h(r)$$

When transformed to Fourier space, this convolution operation translates to a multiplication by the OTF $H(k)$:

$$\mathcal{F}\{I(r)\} = \mathcal{F}\{o(r)\} \times H(k)$$

Here, $\mathcal{F}\{\}$ represents the Fourier Transform. Due to the real-valued nature of the OTF $H(k)$ in widefield microscopy, only amplitude contrast is preserved, leading to the loss of crucial phase information in the image.

To further demonstrate this, a thin sample is assumed. $I(r)$ is the intensity at the camera due to a point source, or coherent illumination and $h(r)$, which is assumed to be real, is the system PSF; r is the spatial coordinates at the camera plane; μ represents the sample absorption, ϕ the phase, and $o(r) = \exp[-\mu(r) + i\phi(r)]$ is the complex transfer function for a thin sample. The intensity at the camera with coherent illumination can be written as:

$$I(r) = \left|\left(h(r)\right) \otimes \exp\left(-\mu(r) + i\phi(r)\right)\right|^2$$

$$I(r) = \left(h(r) \otimes \exp\left(-\mu(r) + i\phi(r)\right)\right)\left(h^*(r) \otimes \exp\left(-\mu(r) - i\phi(r)\right)\right)$$

$$I(r) \cong \left(h(r) \otimes \exp\left(1 - \mu(r) + i\phi(r)\right)\right)\left(h^*(r) \otimes \exp\left(1 - \mu(r) - i\phi(r)\right)\right)$$

$$I(r) \cong (h(r) \otimes 1)(h^*(r) \otimes 1) + \left(h(r) \otimes \left(-\mu(r)\right) + h^*(r)\left(-\mu(r)\right)\right) + i\left(h(r) \otimes \phi(r)\right) - i\left(h^*(r) \otimes \phi(r)\right)$$

As $I(r)$ is calculated, the linear phase term of the OTF is 0, and the phase term, $i\phi(r)$.

$P(u)$ represents pupil function, and $P(0)$ is the zero-frequency component of the pupil function in Fourier space and sets the intensity level of the image background. When $h(r)$ is convolved

with a constant function, it will result in a field that is equal to $P(0)^2$as shown by the equation below.

$$\{h(r)\otimes 1 = P(0)^2\}$$

The OTF in a widefield microscope captures the low-frequency components and serves as a low-pass filter. The above equation can be written as:

$$I(r) = |P(0)^2| - 2\Re\{P^*(0)\big(h(r)\otimes\mu(r)\big)\} + 2\Im\{P^*(0)\big(h(r)\otimes\phi(r)\big)\}$$

In Fourier space:

$$\mathcal{F}\{h(r)\otimes\phi(r) - h^*(r)\phi(r)\} = (\tilde{H}(u) - \tilde{H}^*(-u))\tilde{\phi}(u)$$

In the case of widefield microscopy, the OTF has a real valued PSF, $h(r)$, which exhibits inversion symmetry in frequency space. As a result, the PTF, or the imaginary part of the OTF is effectively 0. This results in the annihilation of the phase information when multiplied with the imaginary part of the Fourier transformed sample field. Only the real valued MTF remains, preserving the amplitude contrast in the final image. With the widefield OTF having a negligible phase component for low spatial frequencies, this makes it difficult to detect phase changes induced by thin samples [25, 27]. As a result, much of the information useful for differentiating structures in thin samples is lost, and the resulting image has poor contrast. To capture the phase information, an asymmetric OTF is essential to break the inversion symmetry and have a non-zero PTF.

2.2 The Weak Object Transfer Function and DPC imaging

When partially coherent, asymmetric illumination is used, such as with DPC, the phase term is non-zero, resulting in a complex OTF. Phase information is encoded into measurable intensity variations as light passes through the sample. This increases contrast for thin samples and allows for the visualization of small variations in the refractive index of different regions within the sample. This is ideal for producing high resolution, contrast enhanced images of biological samples that are typically difficult to image using traditional bright-field microscopy, or require labelling such as fluorescence microscopy [25]. Typically, quantitative phase imaging methods use spatially coherent light. However, partially coherent methods such as DPC allow for 2x better lateral resolution and better optical sectioning and can be performed with a minimum of two images [25].

DPC has several advantages over traditional bright-field and fluorescence microscopy, including its ability to produce high-resolution images of transparent or low-contrast samples, its non-invasive nature, and its ability to provide quantitative phase information about the refractive index of different regions within the sample without the need for staining or labeling [25, 27]. These features make it a valuable tool for a wide range of applications in biology and materials science. In the context of this research, asymmetric illumination with an LED array is used.

In Dr. Waller's paper, "Quantitative differential phase contrast imaging in an LED array microscope" The concept of asymmetric, partially coherent illumination is explored, allowing for phase information to be preserved. Here, the same assumptions are made, where r_c is the spatial coordinates at the camera plane, μ represents the sample absorption, ϕ the phase, $q(r)$ the coherent illumination and $o(r) = exp[-\mu(r) + i\phi(r)]$ is the complex transfer function for a thin sample

[25]. $P(u'')$ is the pupil function, and u'' is the equation of the coordinates at the pupil plane [25]. The intensity at a camera due to a coherent illumination from angle u' can be written as:

$$I(r_c) = \left| \iint \left[\iint q(r)o(r) \exp(-i2\pi r \cdot u'')d^2r \right] P(u'')\exp\left(-2i\pi u'' \cdot r\right)d^2u'' \right|^2$$

When the sample is instead illuminated by an incoherent source, with an intensity distribution of $S(u')$ where u' are the scaled coordinates on the source plane, the intensity at the camera becomes the sum of images due to each point source, and can be written as:

$I(r_c) = |\iint [\iint q(r)o(r) \exp(-i2\pi r \cdot u'')d^2r)]P(u'')\exp\left(-2i\pi u'' \cdot r\right)d^2u''|^2 d^2u'$

For the LED's, the illumination from each LED is approximated by a plane wave at the sample and is written as:

$$q(r) = \sqrt{S(u')}\exp\left(i2\pi u' \cdot r\right)$$

Differential filtering in the pupil plane for each point source produces phase contrast. The asymmetric illumination in the Fourier space creates intensity in real space [25].

After grouping the source and pupil terms into a single transfer function, a 4D or partially coherent transfer function is produced [25]. This is initially nonlinear, but a weak object approximation, $o(r) \approx 1 - \mu(r) + i\phi(r)$ linearizes the problem, resulting in:

$$o(r)o^*(r') \approx 1 - [\mu(r) + \mu(r')] + i[\phi(r) - \phi(r')]$$

The intensity then has three terms, a background, absorption, and phase term. A Fourier transform of both sides is taken, and the intensity spectrum can be written as the sum of all three

terms [25]. The contrast due to absorption and phase are decoupled and linear due to the weak object approximation, and can be written as $H_{abs}(u)$, the optical transfer function for absorption,

$$H_{abs}(u) = -\left[\iint S(u')P^*(u')P(u'+u)d^2u' + \iint S(u')P^*(u')P(u'-u)d^2u'\right]$$

And $H_{ph}(u)$ being the frequency response for the phase, and evaluated independently of each other.

$$H_{ph}(u) = i\left[\iint S(u')P^*(u')P(u'+u)d^2u' - \iint S(u')P^*(u')P(u'-u)d^2u'\right]$$

For the reconstruction of a DPC image and deriving the specific transfer function, at least two images must be taken to obtain a phase contrast image, I^{DPC}. I_T and I_B and are the top and bottom measurements, respectively [25]. Uniformly distributed incoherent point sources in a half circle shape are used to illuminate the sample, where the radius of the circle determines the illumination numerical aperture [25]. Asymmetric illumination results in non-zero transfer functions for both absorption and phase. It also results in a shift in the origin of the spatial frequency spectrum of the object in Fourier space, resulting in a non-zero H_{ph} [25]. The shifted conjugate pupil is no longer symmetric with the original pupil function, enabling capture of the phase information, and leading to a non-zero H_{ph} [25]. In figure one, the asymmetric OTF as it pertains to DPC imaging can be compared to the widefield OTF.

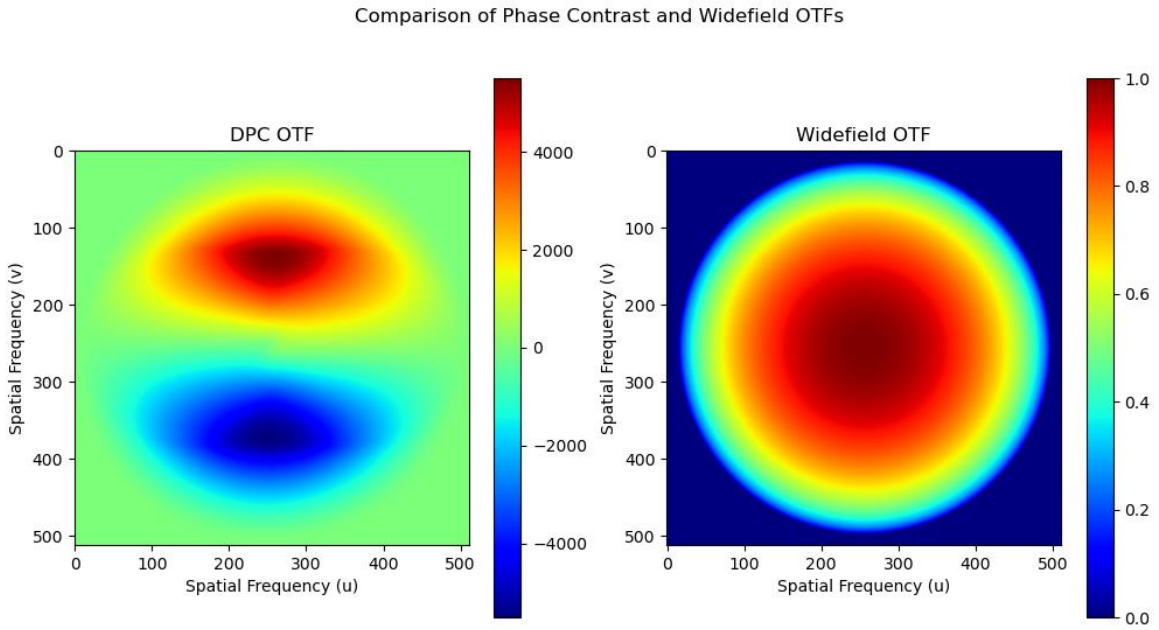


Figure 1 Comparison of Phase Contrast and Widefield OTFs. Here we can see for the DPC imaging system, the WOTF is asymmetric, while the widefield OTF will produce a symmetric OTF that exhibits symmetry, and results in a loss of the phase information when imaging a thin sample or weak object.

To isolate the phase response of the sample asymmetric illumination from one direction can be paired with asymmetric illumination from the other direction. In this way, the background term is canceled, and the phase contrast is better than for single sided asymmetric illumination; furthermore, the quantitative phase can be recovered [25]. The calculation to derive a specific transfer function and to find the phase contrast image I^{DPC} is shown by:

$$I^{DPC}(r_c) = \frac{I_T(r_c) - I_B(r_c)}{I_T(r_c) + I_B(r_c)}$$

Where I_T and I_B represent the top and bottom measurements taken with the incoherent, asymmetric illumination [25].

For the reconstruction of the images, quantitative phase is recovered by deconvolving the DPC image with the calculated transfer function such that:

$$\phi_{tik}(r) = \mathcal{F}^{-1}\left\{\frac{\Sigma_j H_j^*(u) \cdot \tilde{I}_{DPC,j}(u)}{\Sigma_j \left|\mathrm{H}_j(u)\right|^2 + \alpha}\right\}$$

Where $\mathcal{F}^{-1}\{\}$ denotes a Fourier transform, j is the index of DPC measurements, and α is a regularization parameter [25].

2.3 Fourier Ptychographic Imaging

Despite DPC imaging's advantages, the ultimate resolution limit is still governed by the diffraction-limit, resulting in the same resolution as a widefield microscopy system. Fourier Ptychography on the other hand, is concerned with extending the spatial-frequency bandwidth of the imaging system to achieve a higher resolution [23, 24, 27]. The acquired images, consisting of multiple, low-resolution images (usually called "sub-images" or "intensity measurements") are captured under different illumination angles [23, 24]. These illumination angles correspond to different spatial frequency components in the Fourier domain. Thus, for each image captured, the sample is illuminated from a unique angle by turning on a single LED, corresponding to a shift proportional to the angle of illumination.

For each LED, a different area of Fourier space passes through the pupil and enhances the OTF. An iterative phase retrieval algorithm is used to synthesize the full pupil in the Fourier domain, gradually filling in the missing phase information which is lost in imaging, and synthetically increasing the NA to reconstruct the high-resolution image [23, 34]. By combining the low-resolution sub-images taken under various illumination angles, the algorithm effectively expands the bandwidth of the spatial frequency domain and increases the overall resolution of the

reconstructed image [23, 34]. This allows the technique to reconstruct both the amplitude and phase of the sample, resulting in a complex-valued image with enhanced resolution [23, 24, 27]. It provides a higher resolution than DPC imaging but requires more images and time to reconstruct the final image.

CHAPTER 3

METHODS

3.1 Producing the CAR-T cells and Cell Culturing

T cells were obtained from healthy donors, following the ethical guidelines and approval set forth by the Institutional Review Board at the University of Wisconsin-Madison. The T cells were subsequently cultivated. Anti-GD2 CAR-T cells were engineered by introducing CAR transgenes (pSFG.iCasp9.2A.14G2A-CD28-OX40-CD3z, provided by Malcolm Brenner) or mCherry (mCh) via electroporation into the T cells undergoing cultivation. These cultured T cells were sustained in Immunocult-XF expansion medium (StemCell Technologies, Vancouver, BC, Canada). Methods for cultivating and sustaining the T-cells follow procedures as described in "Production and characterization of virus-free, CRISPR-CAR T cells capable of inducing solid tumor regression" [45].

Both varieties of cells, anti-GD2 CAR-T cells and mCherry cells, were cryopreserved at a concentration of 5E6 cells per vial, using a solution consisting of 90% Fetal Bovine Serum (FBS) and 10% Dimethyl Sulfoxide (DMSO). These cells were subsequently transported on dry ice to the University of Georgia for ongoing research. Upon thawing at a temperature of 37°C, both CAR-T and mCherry cells were relocated to culture dishes containing fresh Immunocult-XF expansion medium, enriched with 25µL/mL of CD3/CD28 Activation Serum (StemCell Technologies, Vancouver, BC, Canada). For expanding the cells, the seeding density was adjusted to 1E6 cells/mL every 72 hours, using a renewed supply of Immunocult-XF expansion medium supplemented with 500U/mL of Interleukin-2 (IL-2). These CAR-T cells were received by Dr.

Karumbaiah's lab at the University of Georgia, and samples were shared with Dr. Kner's lab for this project. These cells were received in 35 mm petri dishes, in densities between 5e4 and 8e4 cells per dish/plate.

3.2 Imaging the CAR-T Cells

Anti GD2 CAR-T cells and mCherry CAR-T cells from the same donor are kept in 4ml of ImmunoCult™-XF T Cell Expansion Medium, with Human Recombinant IL-2 (CHO-expressed) before and during imaging. They are stored in an incubator (VWR International, Radnor, Pennsylvania, USA) at 37° C and 5% CO_2. Two dishes were received at a time, one Anti GD2, the other mCherry. For each petri dish imaged, imaging was kept under 45 minutes, taking 8 FPM and 8 DPC images per dish with both a Nikon 4x and 10x objective lens with NAs of 0.13 and 0.30, respectively, resulting in a total of 16 FPM and 16 DPC images for each dish imaged for each day imaged. For the DPC imaging, exposure time is 6 milliseconds for each image, with a total of 4 images being taken. For FPM, exposure time is 16 milliseconds, and 257 images are taken, one for each LED. Every dish was imaged once a day for 3 days. 8 DPC and 8 FPM images are taken per plate, per objective, resulting in 48 total reconstructed images per plate over 3 days. After imaging was complete, the cells were bleached and disposed of in appropriate biohazard containers.

3.3 Image Analysis

After all images are taken, they are reconstructed in a Python program, then loaded into Cell Profiler for segmentation and data collection. Segmentation is first conducted with thresholding using Otsu's method. Predicted object size diameter is altered along with threshold values for intensity and object size to help filter out debris. From Cell Profiler, 155 metrics which are real numbers related to morphology and intensity are output into a CSV file, then the data is

loaded into a Random Forest and KNN machine learning algorithms, and an Autoencoder neural network for analysis. These 155 metrics are listed in Appendix A.

3.4 LED Array Microscope

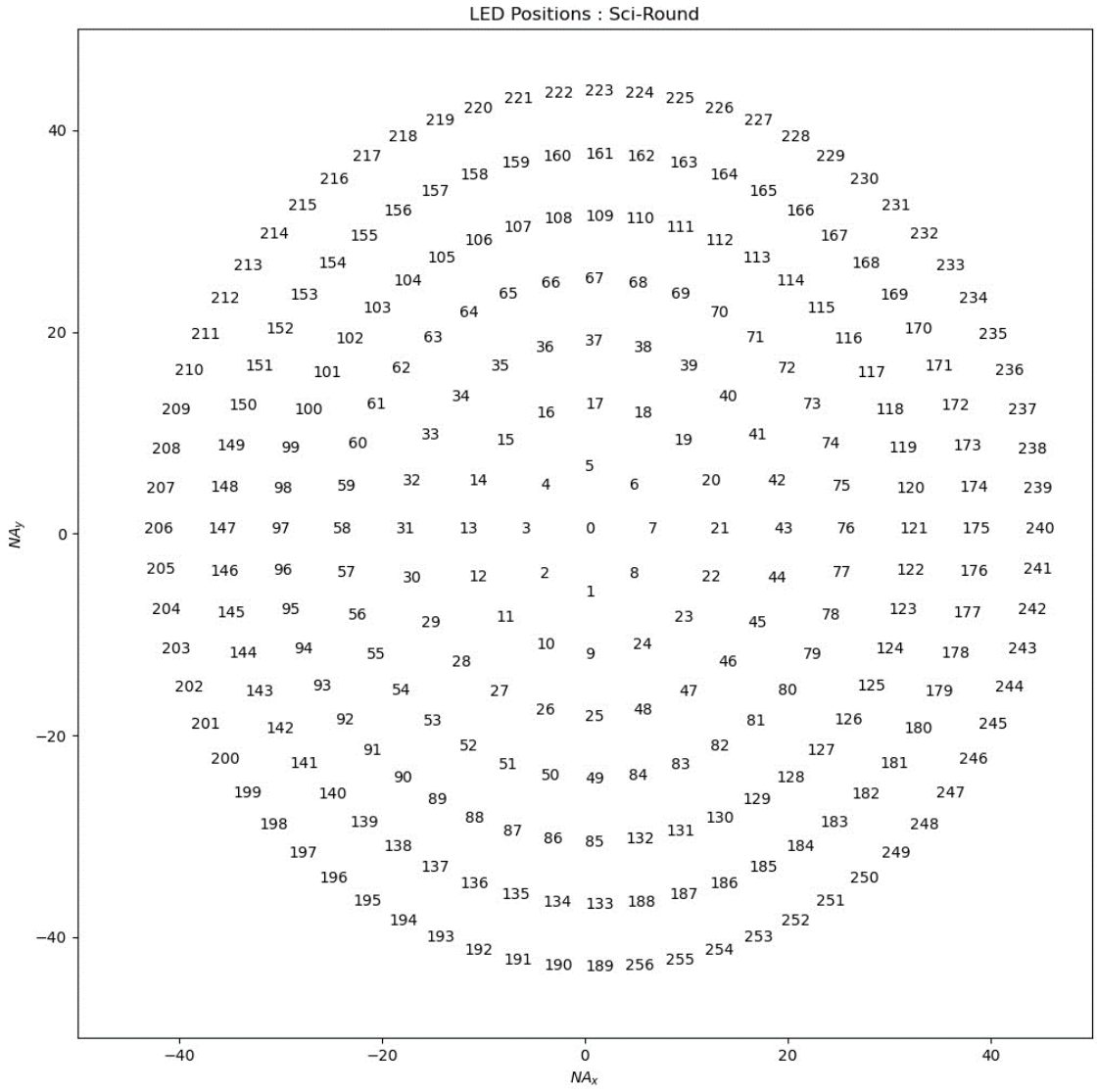


Figure 2. LED Positions: Sci-Round. This is the numbering scheme of the Sci microscopy round LED array, with 257 total LEDs. It has an NA of 0.5 and has a green color channel (257 nm). It uses USB 2.0 a serial interface with the computer.

A PCO panda 4.2 camera with a 16-bit sCMOS sensor and 6.5 x 6.5 μm² pixel size is used for imaging. It can run at up to 40fps, capable of exposure times from 10μs to 5s. Objective lenses used are Nikon Plan Fluor 10x/0.30 NA, and 4x/0.13 NA. A Round LED array from Spectral Coded Illumination Inc. is used to illuminate the sample. It consists of 257 LEDs arranged in concentric circles. For each imaging sequence run, 4 images are taken for DPC using LEDs from the first circle, first taking gradient images from the top, bottom, left, and right, then each of the 257 LEDs are illuminated one at a time sequentially, starting from the first LED, and ending at the final LED in the outermost circle for the FPM images. The maximum NA for the LED array is 0.5. The layout of this LED array can be seen in Figure 2.

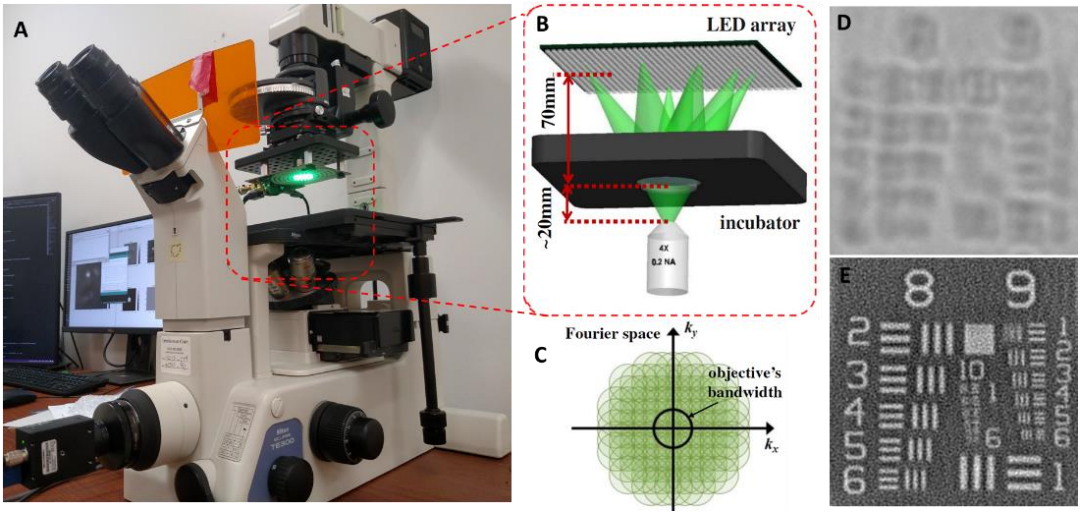


Figure 3. An overview of the DPC/FPM microscope. (A) Nikon TE300 system with modular LED-Array attachment for coded illumination. (B) Close-up diagram of LED-Array, angular illumination and objective optics. (C) Pupil plane view of FPM, where different illumination angles correspond to different circles (L. Tian et. al 2015). (D) Brightfield Image of a phase target (FoV 71x71μm^2). (E) FP reconstruction of (D).

All these components are installed on a Nikon Eclipse TE300, with the LED array set to a working distance of 65mm. Alignment is checked by turning on a sequence of LEDs and taking images for each LED in the sequence, individually illuminating 9 areas of the LED array, LEDs 0, 1, 2, 3, 4, 6, 7, 8. The Fourier transform of each of these images is displayed in a grid sequence, and each displayed k-space image updates every time the LED sequence cycles through. The center of each circle should pass through the diagonal grid lines. Position of the LED array is adjusted manually.

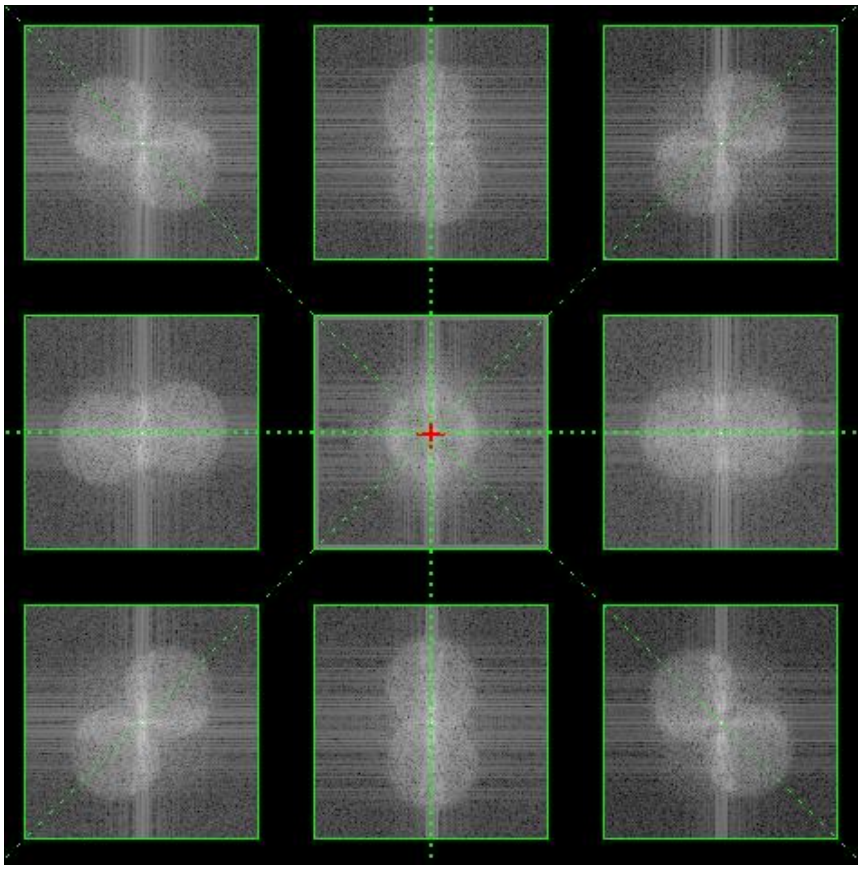


Figure 4. QPI microscope alignment. LEDs From the top left going across each row, 2, 3, 4, 1, 0, 5, 8, 7, and 6 are turned on sequentially. For each image, the Fourier transform is taken is displayed.

3.5 Machine Learning Methods

Autoencoder with a classifier is used to analyze the unlabeled data and predict if the cell is either anti GD2 or mCherry. For the purposes of labelling our data, anti GD2 will be called GD2 positive or GD2+, and mCherry will be called GD2 negative or GD2-. The data is first normalized using a Z score for each data column, with a mean of 0 and a standard deviation of 1. Initially, 80% of the data is used for training and 20% for testing. In this specific model, the latent space has 3 nodes and 32 layers in the hidden space. The imported data is shuffled and split into batches; the shuffling is used to prevent the model from learning any unintentional patterns from the order of the data. Each training epoch goes through the entire data set, and the model's parameters are updated to minimize the sum of reconstruction and classification loss. After the model is trained for 50 epochs, an unseen test data set is passed on to the autoencoder.

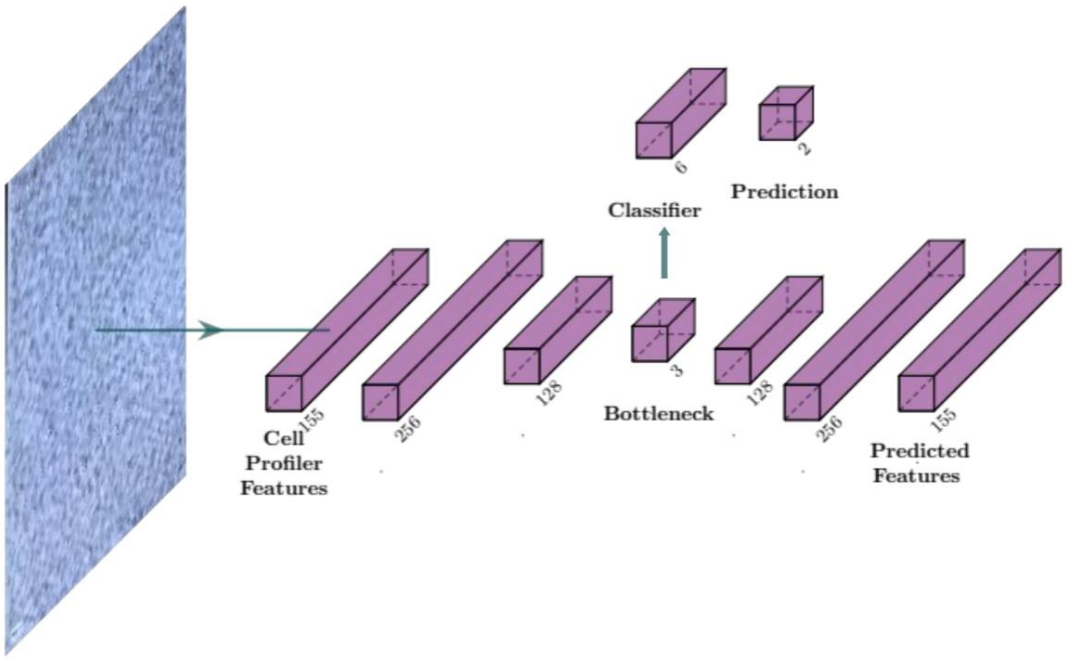


Figure 5. Structure and workflow of the data collection and the autoencoder. This autoencoder features a latent space with 3 nodes.

In the encoder, the first linear layer, the data is transformed into the hidden layers, then is followed by a ReLU (Rectified Linear Unit) activation function that adds non-linearity to the model, enabling it to learn complex patterns. The second linear layer further reduces the dimensionality of the data to the size of the latent space or bottleneck layer. After the encoder has encoded the input data, the lower-dimensional representation is passed through a linear layer to map it to the number of output classes, representing anti GD2 and mCherry CAR T cells. The decoder network then takes this lower-dimensional encoded data and tries to reconstruct the original input data, mirroring the architecture of the encoder but in reverse order, increasing the dimensionality from the latent space to the hidden space, then uses ReLU activation to increase the dimensionality back to the input size of the data.

Mean Squared Error is used for reconstruction loss, measuring how well the autoencoder reconstructs the input data, and cross-entropy loss is used to measure how well the classifier predicts the class of the cells. For training, it runs for 50 epochs, computes the loss based on predicted and actual values, then updates the models' parameters to minimize loss. Through this process, the model learns from the data. The model then examines the test data and computes accuracy, precision, recall, and F1 score to measure the model’s performance.

After the autoencoder, the data is revaluated through KNN and Random Forest machine learning algorithms that are from Python’s sklearn package. We are usiung these three different methods to compare 3 commonly used methods in prior studies for cell classification tasks. Max depth and k are both set to 50. These are set to 50 because in our algorithms, we are also sseeking for the best max depth and best value of k, so these values are set sufficiently high to test for the

best value within each data set. After the data is normalized, it is run through a PCA analysis, then KNN and Random Forest.

For all methods, plots of accuracy, precision, recall, and F1 score, as well as a receiver operating characteristic curve (ROC curve), Area under the ROC Curve (AUC), confusion matrixes, and scatterplots of the classification are produced. For the autoencoder, an additional plot of total loss, classify loss, and reconstruction loss are produced.

3.6 Machine Learning Definitions

Here we define some of the terms used to describe the performances of our machine learning methods.

Accuracy is defined as the proportion of all classifications that are correct. While accuracy can be a useful general measure, it can be misleading when classes are imbalanced or when the costs of different types of errors are significantly different.

Precision quantifies how many of the instances that the model predicted as positive are actually positive. Precision is especially relevant in situations where the cost of a false positive is high.

Recall is calculated as the ratio between the number of Positive samples correctly classified as Positive to the total number of Positive samples. The recall measures the model's ability to detect Positive samples. The higher the recall, the more positive samples detected. It is also known as Sensitivity, True Positive Rate, or Hit Rate. Recall is crucial when the cost of missing a true positive is high.

F1 Score is the harmonic mean of Precision and Recall and tries to balance the two. The F1 score is especially useful in situations where you have an uneven class distribution and both false positives and false negatives are of concern. It takes both false positives and false negatives

into account. A high F1 score indicates a robust model, but it does not guarantee that the model's performance is excellent on both Precision and Recall — it could excel in one and be mediocre in the other.

ROC curve is a graphical representation that plots the True Positive Rate (TPR, also known as Sensitivity or Recall) against the False Positive Rate (FPR, or 1-Specificity) for different threshold values. The curve essentially shows the trade-off between sensitivity and specificity. Each point on the ROC curve represents a different classification threshold, depicting the sensitivity vs. 1-specificity for that threshold.

AUC is the area under the ROC curve and serves as a single scalar value that summarizes the performance of the classifier. The AUC can range from 0 to 1, where a value of 1 indicates a perfect classifier and a value of 0.5 represents a random classifier.

The "Total Loss" is the overall objective function that a neural network aims to minimize during the training process. It is often a combination of multiple loss terms, each serving a specific purpose. In composite models that perform multiple tasks (e.g., reconstruction and classification), the total loss could be a weighted sum or an integration of individual losses like “Recon Loss” and “Classify Loss.” Typically, this is calculated as

$$Total\ Loss = Classification\ Loss + Reconstruction\ Loss$$

The “Classify Loss” is specific to the task of classifying the input data. In neural network models designed for classification tasks, this loss function quantifies how well the model is performing in categorizing the input data into predefined classes. Common classification losses include Cross-Entropy Loss for multi-class problems and Binary Cross-Entropy for binary

classification problems. The aim is to adjust the model parameters to minimize this loss, thereby increasing the model's classification accuracy. For a Cross-Entropy Loss, y_{ic} is the ground truth label, $\hat{y}_{ic}$ is the predicted distribution, or output of the network. N is the batch size, and C is the number of classes. It can be written as:

$$Classification\ Loss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic})$$

In models like autoencoders, the "Reconstruction Loss" measures the quality of the reconstructed output compared to the original input. The goal of minimizing this loss is to make the reconstructed data as close as possible to the original data. Common reconstruction losses include Mean Squared Error (MSE) for continuous data and Binary Cross-Entropy for binary data. For MSE, X is the input and $\hat{X}$ is the reconstruction.

$$Reconstruction\ Loss = \frac{1}{N}\sum_{i=1}^{N}(X_i - \hat{X}_i)^2$$

CHAPTER 4

RESULTS

4.1 Imaging and Segmentation

For each FPM image taken with a 4x objective, the total field of view is 1626.60 x 1626.60 µm, or 10160 x 10160 pixels with a resolution of 0.3826 µm. For the images taken with a 10x objective lens, the total field of view is 650.64 x 650.64 µm, 6096 x 6096 pixels, and has a resolution of 0.3096 µm. Figure 6 shows the full field of view (FoV) of the FPM reconstructions.

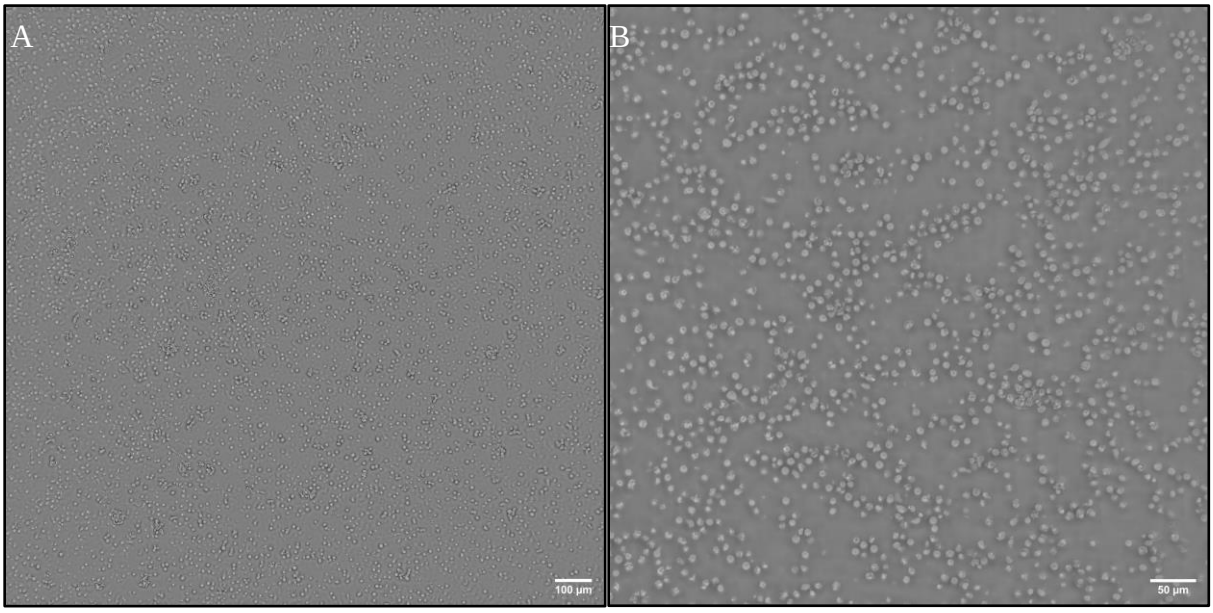


Figure 6: Full field of view FPM reconstructions. (A) FPM reconstruction taken with a 4x objective lens with a FoV of 1626.60 x 1626.60 µm and a resolution of 0.3826 µm. (B) FPM reconstruction taken with a 10x objective with a FoV 650.64 x 650.64 µm, and a resolution of 0.3096 µm.

For DPC, the images are 2048x2048 pixels, and have the same respective fields of view. Resolution for the DPC 4x and 10x images is 2.046 µm. and 0.887 µm , respectively. For the full field of view DPC images with a 4x objective, ~6000 cells can be seen. For imaging and segmentation, each image has had anywhere between ~500 and ~6000 cells. For the 10x images, ~1500 cells or less have been seen in a single image. These numbers can increase or decrease depending on the density of cells in the petri dish, or viability of the cells over several days of imaging. The number of cells captured in each image decreased over time as there were fewer viable cells. An example of DPC images as well as cell segmentation can be found in figures 7 and 8.

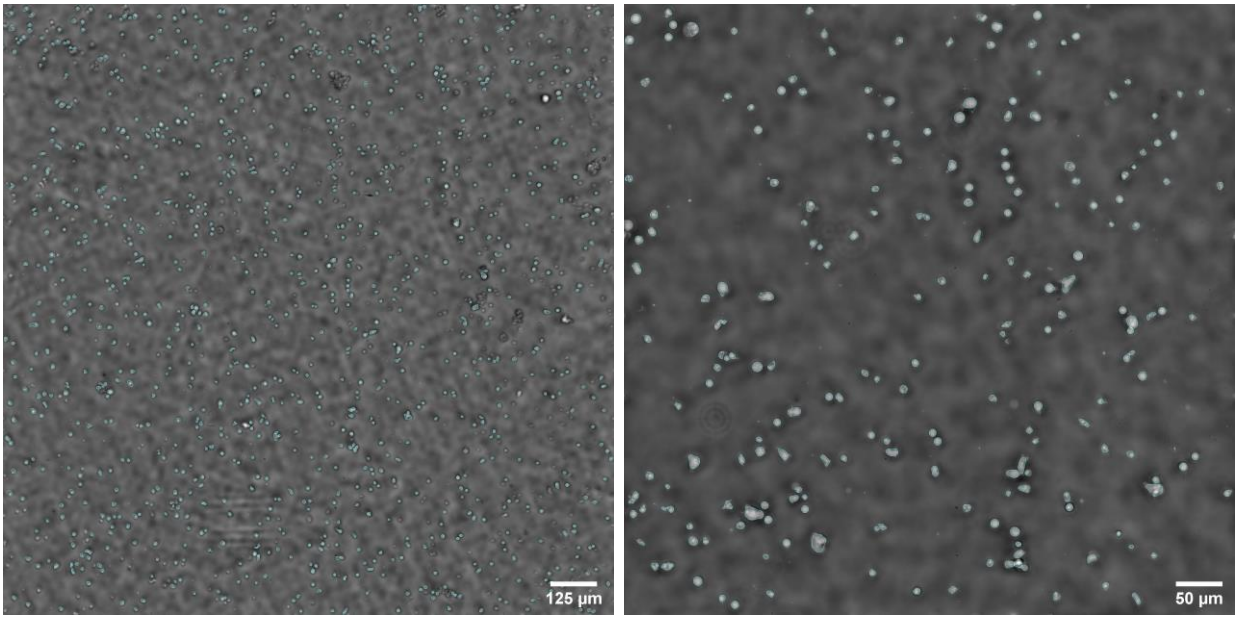


Figure 7. Sample segmentation of 4x and 10x DPC images. Successful segmentation on (A) 4x DPC and (B) 10x DPC images. Both are GD2+ cells from the same sample.

Segmentation with CellProfiler was evaluated as each batch of images was being processed, manually checking each image for appropriate segmentation and making adjustments

to the CellProfiler pipeline as necessary until most of the cells were being segmented. For each pipeline, changes in the average diameter of the object being detected is changed basted on whether or not the image was taken with a 4x or 10x objective, since the size of the cell in pixels changes between these image sets with increased resolution. Minor changes in filtering based on intensity and object size also is modified depending on the presence of debris in the samples. An example of the pipeline structure can be found in Appendix B. Cells touching the border of the images were excluded. No ground truth count for cells in each image was established, so the percentage of cells correctly segmented and counted were not calculated, however some estimates regarding missed cells per image were made by counting the missed cells in a few individual images. In the 4x DPC images, 150- 200 cells were typically missed in segmentation, while with the 10x images, fewer than 20 cells were missed. Segmentation was only performed on DPC data.

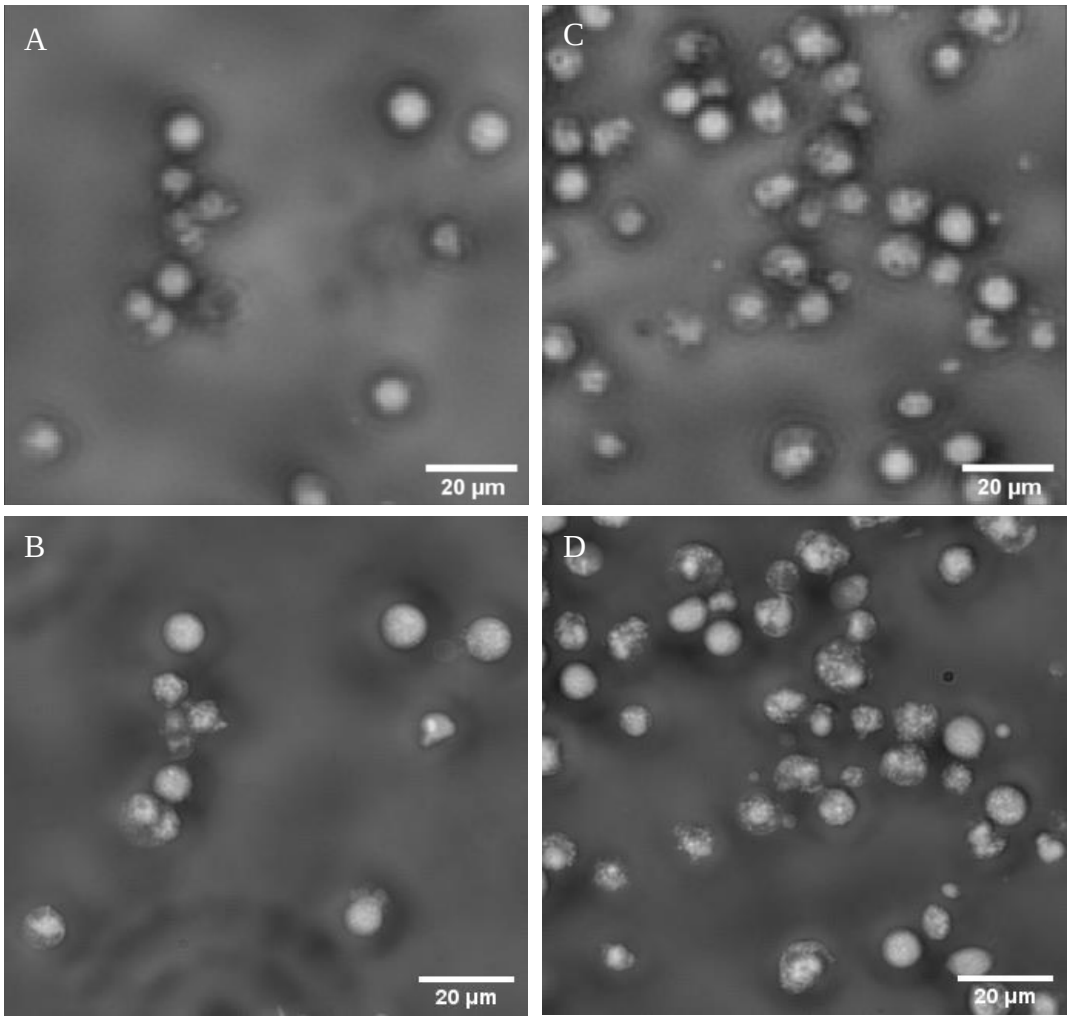


Figure 8. Comparison of 4x and 10x DPC GD2+ and GD2- Cells. (A) is the 4x DPC image of GD2+ cells, (B) the 10x image of the same cells. (C) is the 4x image of GD2- cells, and (D) the 10x image of the same cells.

Qualitatively assessing the images, the DPC images appeared to have greater contrast between the foreground and background of the images while the FPM images tended to lack good contrast for segmentation. For the this analysis, we only analyzed the DPC images, with plans to evaluate the FPM results after modifications to the reconstruction code.

4.2 Machine Learning Results

For evaluating the three machine learning methods, initially, we received 3 total plates of GD2 positive and 3 of GD2 negative CAR-T cells for a total of 6 combined plates. These plates were imaged across three days, except for the final two plates, which were only image for two days due to insufficient media on the third day. In the machine learning results, we looked at precision, accuracy, recall, F1, and AUC scores for both 10x and 4x magnifications over 3 days of imaging to see which model was the most consistent over multiple days of imaging. Only the DPC images were evaluated for the scope of this project.

We first evaluated the aggregate data from day 1 of imaging, when the cells were at their healthiest. The aggregate data consists of all the images from all plates taken with a particular objective on the same day of imaging. For the 4x DPC training data, Random Forest outperformed both KNN and Autoencoder, with accuracy, precision, recall, and F1 scores of 100%. The autoencoder was slightly lower, with accuracy, precision, recall, and F1 scores of 95.61%, 94.67%, 94.98%, and 94.82%. However, when it came to the performance of the test data, the autoencoder performed best, with an AUC score of 0.98. while Random Forest and KNN had AUC scores of 0.89 and 0.88. On the test data, for Random Forest, accuracy precision, recall, and F1 were 81.92%, 84.82%, 90.38%, and 87.51%, respectively. KNN was significantly lower, with most metrics scoring below 90%. Figure 9 shows the plots of each of these, both testing and training data. Figure 10 shows the ROC curves for both testing and training, and Figure 11 shows the confusion matrices for this data set.

Figure 9. Aggregate 4x Day 1 Training and Testing Data for all methods. The first row of graphs displays the training data results of accuracy, precision, recall, and F1. The second row shows these same performances on the test data set. Random Forest outperformed KNN and autoencoder on the training data, however, autoencoder outperformed both methods on the testing data for all the evaluated metrics.

For the ROC curves in Figure 10, AUC scores were the highest for autoencoder, and lowest on KNN on the test data. While showing the trade off between recall and the false positive rate at specific thresholds, we can see how close to random a classifier is with the AUC score, and examine the classifier as a whole. AUC is particularly useful when the data set is imbalanced, such as with the data for this project.

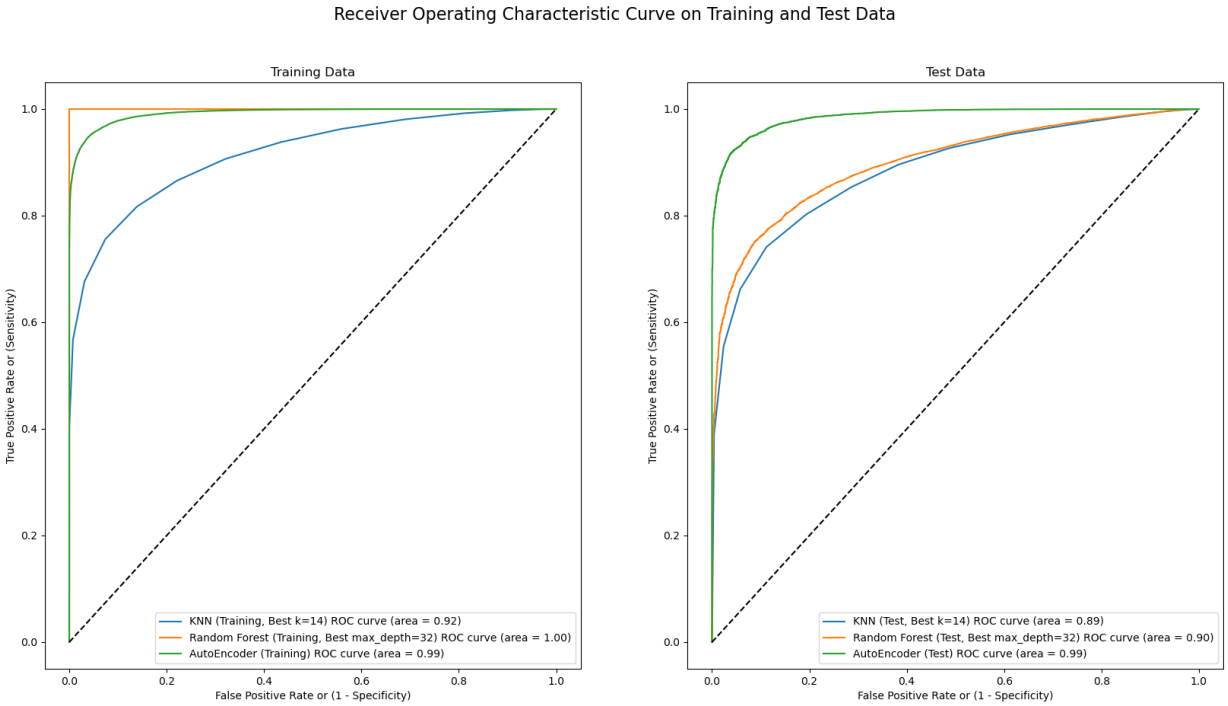


Figure 10. ROC Curve on Aggregate 4x Day 1 Training and Testing Data. Autoencoder had the best AUC score of the three methods on the test data.

When examining the confusion matrices in Figure 11, we see that KNN produced a significant amount of false positives in the training and testing data set, whereas Random Forest and Auto encoder did not have huge imbalances of false positives and false negatives in the training data set, Random Forest had nearly double the amount of false positives in the test data set. This high number of false positives results in a lower precision. F1 is a useful metric when there is a class imbalance. When considering the best model, finding one that best balances the scores of precision and recall is necessary when aiming to increase the number of true positives and reduce the number of false positives. Accuracy, while useful, can be misleading when classes are heavily imbalanced.

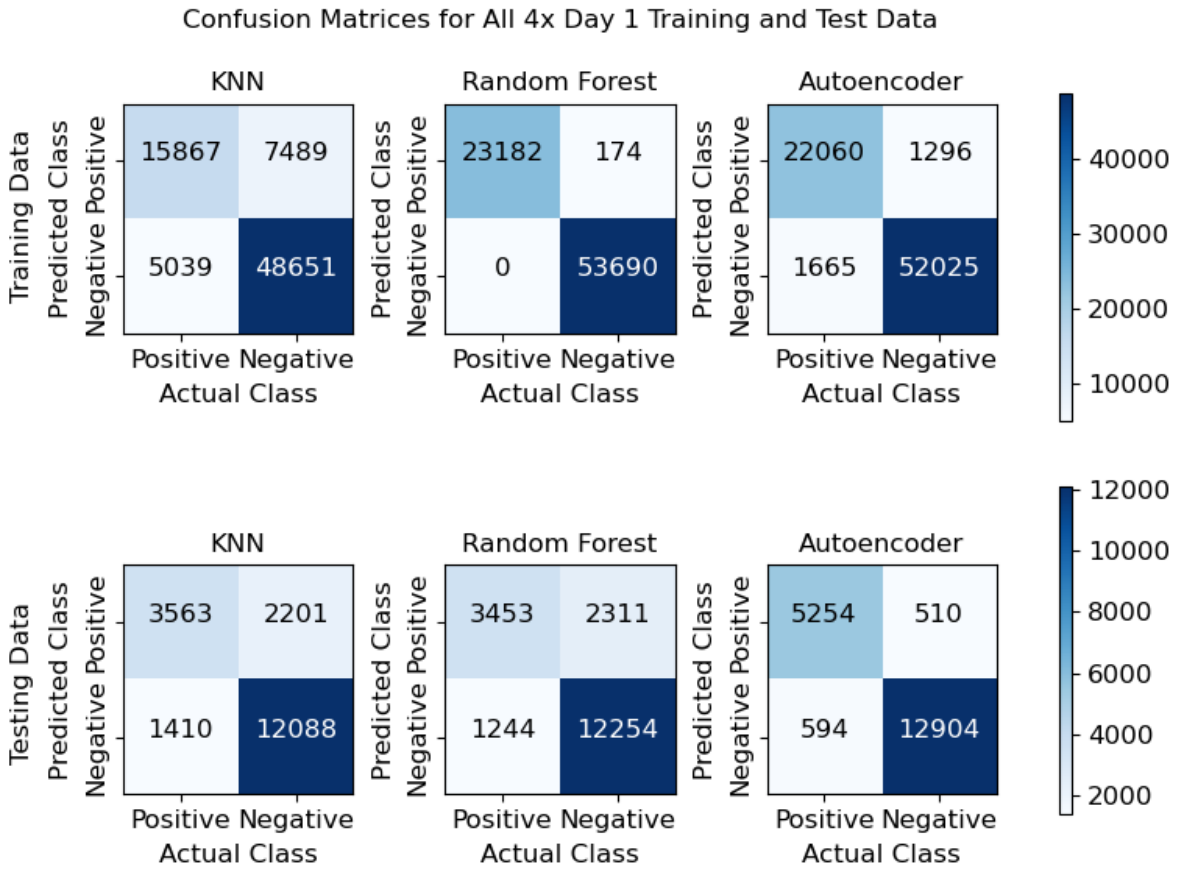


Figure 11. Confusion Matrices for Aggregate 4x Day 1 Training and Testing Data. Row one shows the confusion matrices on the training data set, while the second row shows the results for the test data set. While Random Forest performed well on the training data, for the test data set, Random Forest and KNN had a high number of false positives, while autoencoder reduced the amount of both false positives and false negatives.

For the 10x data in Figure 12, similar trends were observed for the training data set, however the test data set performed differently. The autoencoder performed better on accuracy; however, it had worse performance on precision and recall, which are good indicators of the reduction of false positives and negatives. The F1 score was higher. The AUC scores on the 10x

data for KNN and Random Forest were slightly better than the 4x data at 0.90 and 0.91, but the autoencoder was lower at 0.96.

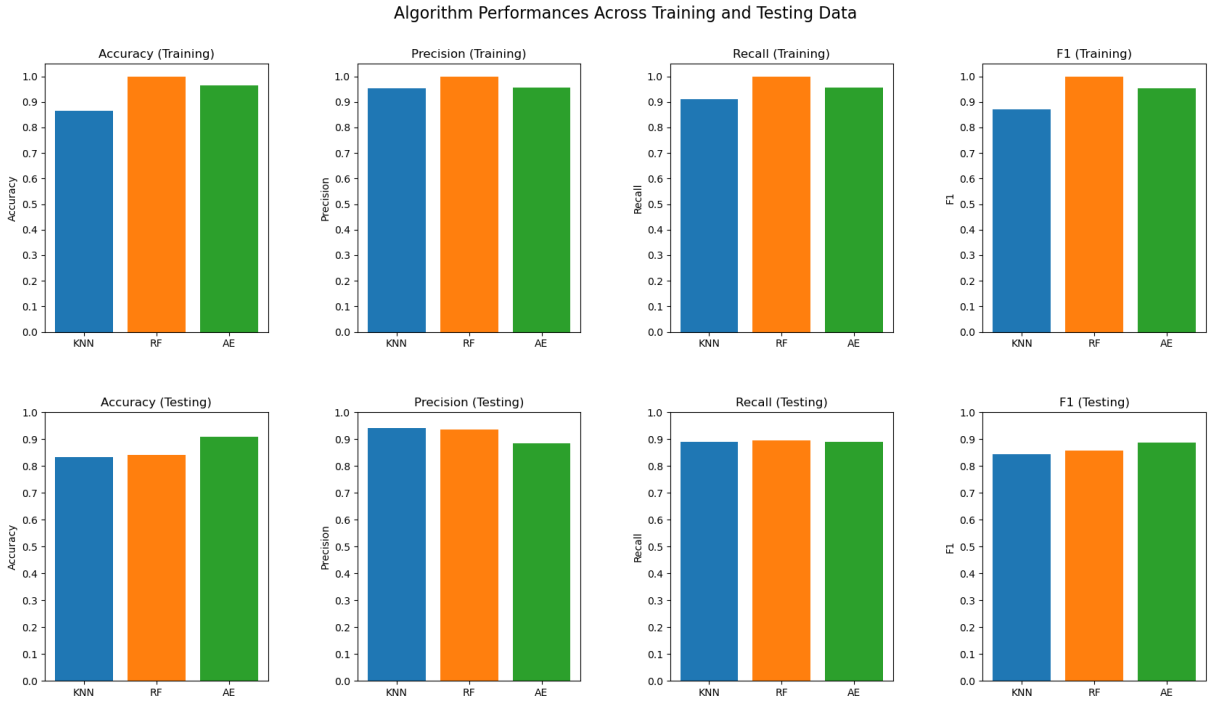


Figure 12. Algorithm Performances Across Aggregate 10x Day 1 Training and Testing Data. Accuracy and F1 is high for the autoencoder, but precision and recall are lower, indicating there may be more false positives and false negatives.

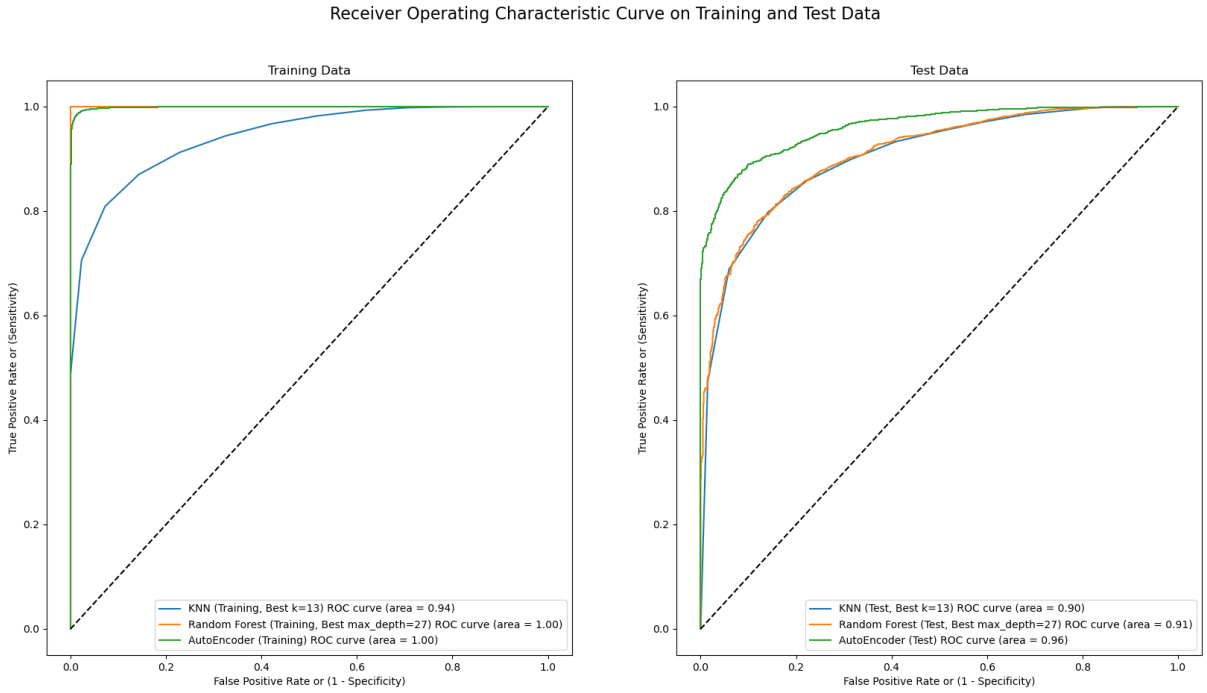


Figure 13. ROC Curve on Aggregate 10x Day 1 Training and Testing Data. For the test data, Autoencoder had the highest AUC score at 0.96. Random Forest and KNN had results of 0.91 and 0.90.

In Figure 14, for the autoencoder, there is a higher number of false negatives, but fewer false positives in both the training and test data. This leads to lower precision. Due to the low number of false positives, a higher accuracy is also obtained. Both Random Forest and KNN had high numbers of false positives for the training data evaluation.

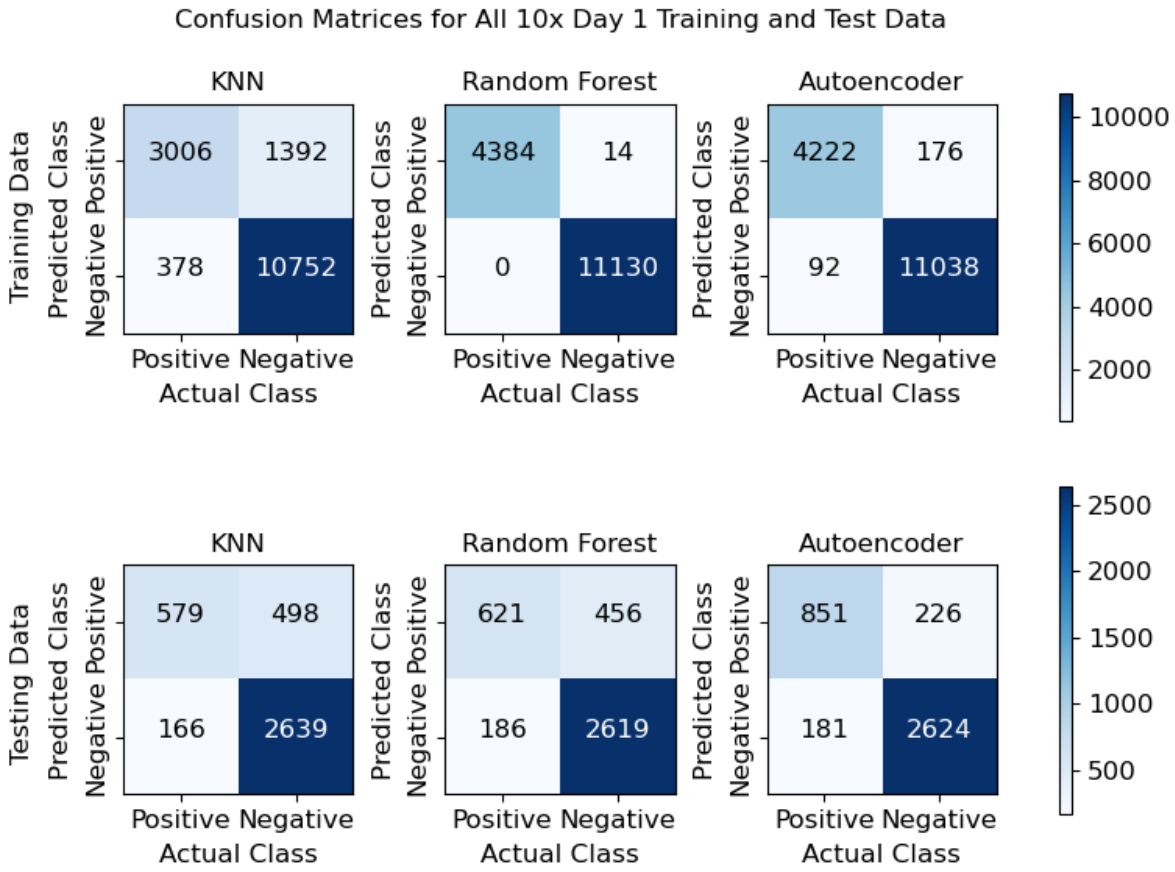


Figure 14. Confusion Matrices for Aggregate 10x Day 1 Training and Testing Data. The autoencoder shows a higher proportion of false negatives for the test data, whereas KNN and Random Forest have a much higher proportion of false positives.

To further confirm the autoencoder was the best model over multiple days of imaging, the results of each machine learning method were compared for aggregate data over three days. With the autoencoder model performing the best for both 4x and 10x data sets for the first day of imaging, and having the highest AUC score, this is the primary method we moved forward with. Tables 1 through 6 show the aggregate data for the autoencoder as the cells were evaluated over 3 days, with the third day only being an aggregate of the first two sets of plates since the last set was

not imageable for the third day. The autoencoder still outperforms KNN and Random Forest. The tables for KNN and Random Forest can be found in Appendix D.

Table 1: DPC 4x Autoencoder Aggregate Data Over 3 days

	Day 1		Day 2		Day 3	
	Training	Testing	Training	Testing	Training	Testing
Accuracy	95.61	93.89	96.61	93.70	100	98.19
Precision	94.67	92.69	96.72	93.78	100	97.34
Recall	94.98	92.75	96.34	93.29	100	97.01
F1	94.82	92.72	96.51	93.51	100	97.18
AUC	0.99	0.99	0.99	0.98	1	0.99

Table 2: DPC 10x Autoencoder Aggregate Data Over 3 days

	Day 1		Day 2		Day 3	
	Training	Testing	Training	Testing	Training	Testing
Accuracy	98.44	90.34	98.57	84.27	100	92.82
Precision	98.15	88.16	98.34	83.19	100	92.39
Recall	98.00	87.56	98.56	83.45	100	92.62
F1	98.07	85.85	98.45	83.32	100	92.50
AUC	1	0.96	0.99	0.93	1	0.98

Noting the discrepancy between the 4x and 10x evaluations, we further examined the plates of data as individual sets, meaning, the first two plates on a single day were evaluated, the second

set on a singular day were examined as a separate set, etc. This was done for each day the plates were imaged. These plates are numbered 5, 6, 7, 8, 9, 10, the numbering starting from when we began to receive both GD2 positive and GD2 negative plates. 5, 7, 9, are GD2 positive and 6, 8, 10 are GD2 negative. A “pair” of plates would be 1 GD2+ and one GD2- plate, such as 5 and 6, 7 and 8, or 9 and 10.

Evaluating just plates 5 and 6 on the first day, initially, all methods have high performance, including the 10x data, as seen in Figure 15 which compares the 4x and 10x performance on the testing data set for plates 5 and 6. In Figure 16, comparing plates 7 and 8, we see start to see a significant decrease in performance for the 10x data. Table 3 compares the 4x autoencoder data and AUC scores for plates 5, 6, 7 and 8, and Table 8 for the 10x data. In Table 4, despite the lower performance on accuracy, precision, F1, and recall, AUC is still high. When examining the confusion matrices for only the 10x data in plates 7 and 8 in Figure 17, we see slightly more false positives than false negatives, however with KNN and RF, there are a greater number of false positives. The performance of the autoencoder is consistent with that of the 4x data, however, with more instances of falsely classifying data in a greater proportion than the 4x data. This behavior is also seen in plates 9 and 10, which can be found in Appendices C and D.

Table 3: DPC 4x Autoencoder Test Data Over 3 Days, Plates 5, 6, 7, 8

	Day 1		Day 2		Day 3	
	5 & 6	7 & 8	5 & 6	7 & 8	5 & 6	7 & 8
Accuracy	99.42	97.46	97.30	95.25	98.88	95.26
Precision	98.88	97.64	96.24	95.17	97.44	95.25
Recall	98.77	96.92	95.64	95.21	98.02	95.21
F1	98.82	97.26	95.94	95.19	97.73	95.23
AUC	0.99	0.99	0.99	0.99	0.99	0.99

Table 4: DPC 10x Autoencoder Test Data Over 3 Days, Plates 5, 6, 7, 8

	Day 1		Day 2		Day 3	
	5 & 6	7 & 8	5 & 6	7 & 8	5 & 6	7 & 8
Accuracy	99.87	86.02	100	76.52	98.78	90.43
Precision	99.72	86.12	100	76.76	98.86	90.29
Recall	99.82	85.72	100	76.11	98.50	90.46
F1	99.77	85.86	100	76.21	98.67	90.36
AUC	0.99	0.93	1	0.83	0.99	0.96

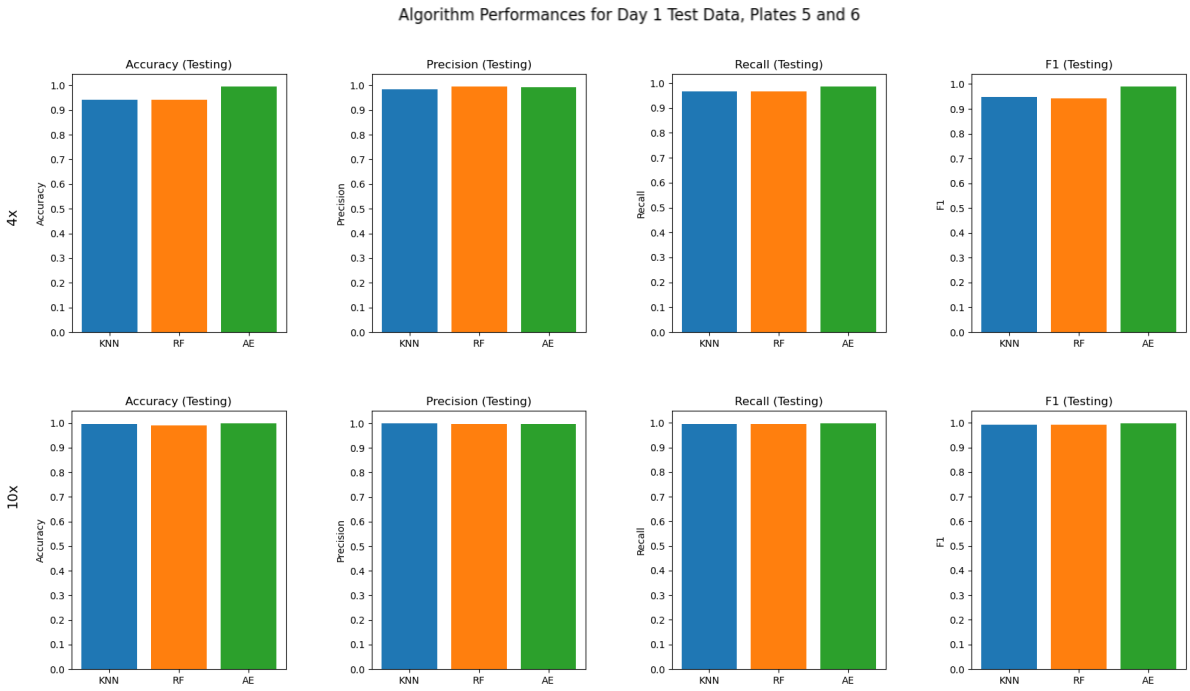


Figure 15. Algorithm Performances for Day 1 Training and Testing Data, Plates 5 and 6. For Plates 5 and 6, all methods across both 4x and 10 had high performances. For 4x, accuracy, precision, recall, and F1 were all > 98%, and for 10x, > 98%.

Figure 16. Algorithm Performances for Day 1 Training and Testing Data, Plates 7 and 8. For the autoencoder 4x, accuracy, precision, recall, and F1 were > 98%, whereas for the 1x data, it was <90%.

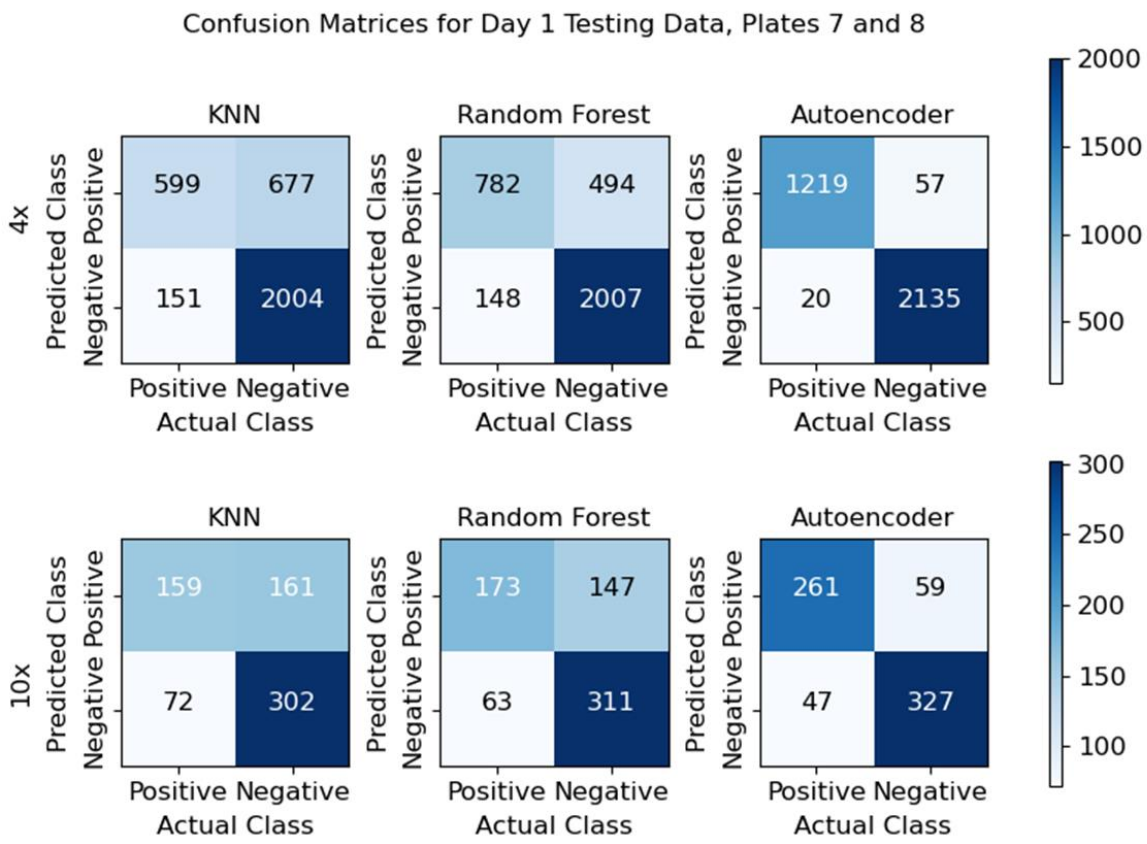


Figure 17. Confusion Matrixes for Day 1 Testing Data, Plates 7 and 8. While there are fewer false positives compared to KNN and Random Forest methods for 10x data, there are more incorrect classifications for the 10x autoencoder data compared to the 4x data.

To further evaluate this, we examined the total, reconstruction, and classification losses from the autoencoder for this data set and the aggregate data sets. Figure 18 shows the losses for the autoencoder model for only plates 7 and 8. We see minimal to no overfitting over 30 epochs. However, for the 10x data, we see overfitting start by 30 epochs. For the other datasets, available in Appendix C, we see overfitting primarily with the 10x data sets outside of plates 5 and 6.

Tables of each pair of plates being evaluated over 3 days, as well as more graphs for performances, ROC, autoencoder classification, reconstruction, and total loss, and confusion matrices for both training and testing data can be found in Appendices C and D.

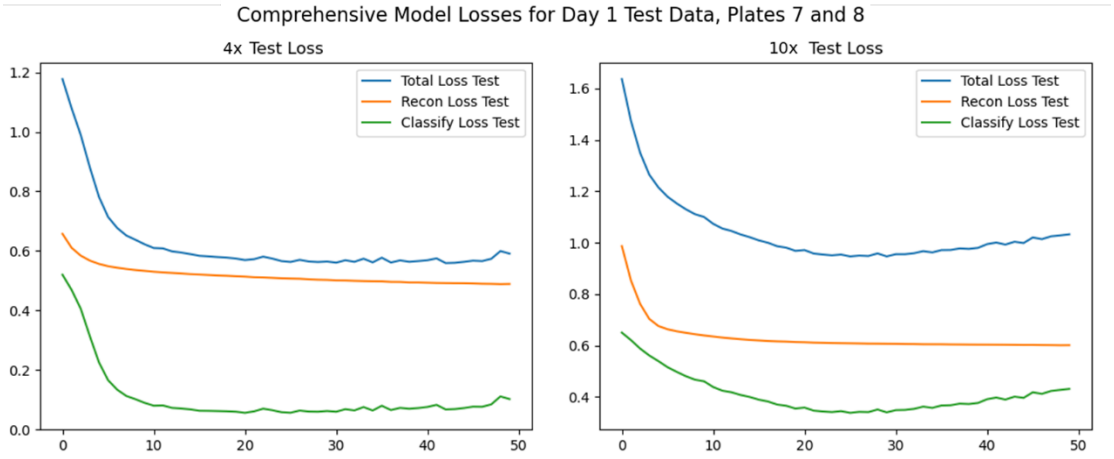


Figure 18. Autoencoder Model Losses for Day 1 Test Data, Plates 7 and 8. The 10x data showed signs of overfitting the data after 30 epochs of training.

CHAPTER 5

DISCUSSION AND CONCLUSION

5.1 Summary of Results

In this study we were able to demonstrate that across multiple days of imaging, and multiple plates of CAR-T cells, we can classify whether or not they were GD2 positive or negative with the machine learning methods KNN, RF, and autoencoder. The autoencoder performed better overall, with consistently higher AUC scores, and higher precision. The 4x data tended to perform better, with aggregate assessments of the data over 3 days showing accuracy, precision, recall, and F1 values of 92% or greater, and individual data batches typically having accuracies of 95% or greater. AUC scores for the 4x aggregate data were > 0.98 and >0.93 for the 10x aggregate data over 3 days of imaging.

The 10x data, however, per batch and aggregate evaluations, performed worse than the 4x data, with aggregate evaluations by day having accuracy, precision, F1, and recall scores of as low as 83%, and as low as 76% on some individual batches. The first set of plates, labeled dishes 5 and 6 in the supplementary information, had consistent performance with high accuracy across both objectives and all three days, plates 7, 8, 9, and 10 all showed a decrease in performance, most drastically with the images taken with a 10x objective. Additionally, these datasets for the 10 images exhibited overfitting. These issues may be due to class imbalances, which can effect the training of the machine learning models, or due to errors in alignment of the LED array or leveling of the microscope, which can affect image quality and segmentation.

Alternatively, DPC is sensitive to sample thickness, and since higher NA objectives have a smaller depth of focus, images taken with the 10x objective may be more sensitive to focal drift from issues in stage levelling and microscope alignment. CAR-T cells are non-adherent cultures and can freely move in the media, which could also affect the DPC imaging.

Overall, the KNN and RF algorithms performed worse across both objectives, albeit significantly worse on the 10x data. KNN and RF need more data to train on, and 10x data, due to its smaller field of view, had smaller data sets to work with. For the 4x data, Random Forest was the second-best method, with KNN performing the worst overall.

5.2 Future Directions

For this project, we would like to further evaluate why the data behaved the way it did with the autoencoder. With the 4x data performing better than the higher resolution images, knowing why would be useful in further development of a high throughput imaging system for quantitatively assessing CAR-T cells. A series of validation experiments with fluorescent imaging will also be conducted to further establish the use of phase imaging with the autoencoder to classify cells as Anti GD2 and mCherry, and eventually to identify successfully transfected T-cells.

While FPM images were not analyzed with machine learning in this project, we would like to improve the reconstruction algorithm to utilize the DPC images, combining the high frequencies of FPM images and low frequency data from the corresponding DPC images to improve contrast and resolution of these images. It is known “without DPC initialization, the low-frequency components of the phase are not well recovered” [23]. This results in a high-pass-filtering effect on the re- constructed phase. By changing our reconstruction code to utilize DPC initialization, low frequencies can be recovered correctly.

For segmentation, we would like to move away from using CellProfiler, and start using the image data and pixel values directly or segment the cells using deep learning methods before assessing it with the autoencoder. With this, we hope to be able to look further at assessing potency or other CQA's with quantitative phase imaging.

Overall, with all these considerations, we would like to implement FPM and DPC imaging methods on a smaller microscope, that could easily be transported between labs, used in incubators, in conjunction with flow cytometry, or bioreactors. With the development of this smaller microscope and moving every part of the imaging, reconstruction, segmentation, image analysis, and classification into python, we could have a singular, easy to implement process for the imaging and evaluation of CAR-T cells.

REFERENCES

1. National Academies of Sciences, Engineering, and Medicine; Health and Medicine Division; Board on Health Sciences Policy; Forum on Regenerative Medicine; Beachy SH, Schumm SN, Nicholson A, editors. Training the Regenerative Medicine Workforce for the Future: Proceedings of a Workshop—in Brief. Washington (DC): National Academies Press (US); 2023 May 11. Available from: https://www.ncbi.nlm.nih.gov/books/NBK592454/ doi: 10.17226/27013
2. Almåsbak H, Aarvak T, Vemuri MC. CAR T Cell Therapy: A Game Changer in Cancer Treatment. J Immunol Res. 2016;2016:5474602. doi: 10.1155/2016/5474602. Epub 2016 May 19. PMID: 27298832; PMCID: PMC4889848.
3. Biomanufacturing and Synthetic Biology [Internet]. Centers for Disease Control and Prevention; 2019 [cited 2023 Sept 21]. Available from: https://www.cdc.gov/niosh/topics/advancedmnf/biomnf.html
4. Holland I, Davies JA. Automation in the Life Science Research Laboratory. Front Bioeng Biotechnol. 2020 Nov 13;8:571777. doi: 10.3389/fbioe.2020.571777. PMID: 33282848; PMCID: PMC7691657.
5. Saha K, Roy K. Integrating United States Biomanufacturing Across Vaccines and Therapeutics. NAM Perspect. 2021 Apr 19;2021:10.31478/202104e. doi: 10.31478/202104e. PMID: 34532694; PMCID: PMC8406570.
6. Nogueira DES, Cabral JMS, Rodrigues CAV. Single-Use Bioreactors for Human Pluripotent and Adult Stem Cells: Towards Regenerative Medicine Applications.

Bioengineering (Basel). 2021 May 17;8(5):68. doi: 10.3390/bioengineering8050068. PMID: 34067549; PMCID: PMC8156863.

7. Zhang C, Durer S, Thandra KC, et al. Chimeric Antigen Receptor T-Cell Therapy. [Updated 2022 Oct 3]. In: StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing; 2023 Jan-. Available from: https://www.ncbi.nlm.nih.gov/books/NBK537294/
8. Vairy S, Garcia JL, Teira P, Bittencourt H. CTL019 (tisagenlecleucel): CAR-T therapy for relapsed and refractory B-cell acute lymphoblastic leukemia. Drug Des Devel Ther. 2018 Nov 12;12:3885-3898. doi: 10.2147/DDDT.S138765. PMID: 30518999; PMCID: PMC6237143.
9. Majzner RG, Ramakrishna S, Yeom KW, Patel S, Chinnasamy H, Schultz LM, Richards RM, Jiang L, Barsan V, Mancusi R, Geraghty AC, Good Z, Mochizuki AY, Gillespie SM, Toland AMS, Mahdi J, Reschke A, Nie EH, Chau IJ, Rotiroti MC, Mount CW, Baggott C, Mavroukakis S, Egeler E, Moon J, Erickson C, Green S, Kunicki M, Fujimoto M, Ehlinger Z, Reynolds W, Kurra S, Warren KE, Prabhu S, Vogel H, Rasmussen L, Cornell TT, Partap S, Fisher PG, Campen CJ, Filbin MG, Grant G, Sahaf B, Davis KL, Feldman SA, Mackall CL, Monje M. GD2-CAR T cell therapy for H3K27M-mutated diffuse midline gliomas. Nature. 2022 Mar;603(7903):934-941. doi: 10.1038/s41586-022-04489-4. Epub 2022 Feb 7. PMID: 35130560; PMCID: PMC8967714.
10. Center for Drug Evaluation and Research. Current good manufacturing practice (CGMP) regulations [Internet]. FDA; [cited 2023 Sept 13]. Available from: https://www.fda.gov/drugs/pharmaceutical-quality-resources/current-good-manufacturing-practice-cgmp-regulations

11. Davila ML, Brentjens R, Wang X, Rivière I, Sadelain M. How do CARs work?: Early insights from recent clinical studies targeting CD19. Oncoimmunology. 2012 Dec 1;1(9):1577-1583. doi: 10.4161/onci.22524. PMID: 23264903; PMCID: PMC3525612.
12. Sterner, R.C., Sterner, R.M. CAR-T cell therapy: current limitations and potential strategies. Blood Cancer J. 11, 69 (2021). https://doi.org/10.1038/s41408-021-00459-7
13. Sadelain M, Brentjens R, Rivière I. The basic principles of chimeric antigen receptor design. Cancer Discov. 2013 Apr;3(4):388-98. doi: 10.1158/2159-8290.CD-12-0548. Epub 2013 Apr 2. PMID: 23550147; PMCID: PMC3667586.
14. Almåsbak H, Aarvak T, Vemuri MC. CAR T Cell Therapy: A Game Changer in Cancer Treatment. J Immunol Res. 2016;2016:5474602. doi: 10.1155/2016/5474602. Epub 2016 May 19. PMID: 27298832; PMCID: PMC4889848.
15. Yonghong Li, Yan Huo, Lei Yu, Junzhi Wang, Quality Control and Nonclinical Research on CAR-T Cell Products: General Principles and Key Issues, Engineering, Volume 5, Issue 1, 2019, Pages 122-131, ISSN 2095-8099, https://doi.org/10.1016/j.eng.2018.12.003. (https://www.sciencedirect.com/science/article/pii/S2095809918308786)
16. Si X, Xiao L, Brown CE, Wang D. Preclinical Evaluation of CAR T Cell Function: In Vitro and In Vivo Models. Int J Mol Sci. 2022 Mar 15;23(6):3154. doi: 10.3390/ijms23063154. PMID: 35328572; PMCID: PMC8955360.
17. Peinelt A, Bremm M, Kreyenberg H, Cappel C, Banisharif-Dehkordi J, Erben S, Rettinger E, Jarisch A, Meisel R, Schlegel PG, Beck O, Bug G, Klusmann JH, Klingebiel T, Huenecke S, Bader P. Monitoring of Circulating CAR T Cells: Validation of a Flow Cytometric Assay, Cellular Kinetics, and Phenotype Analysis Following Tisagenlecleucel.

Front Immunol. 2022 Mar 2;13:830773. doi: 10.3389/fimmu.2022.830773. PMID: 35309367; PMCID: PMC8926389.

18. Kiesgen S, Messinger JC, Chintala NK, Tano Z, Adusumilli PS. Comparative analysis of assays to measure CAR T-cell-mediated cytotoxicity. Nat Protoc. 2021 Mar;16(3):1331-1342. doi: 10.1038/s41596-020-00467-0. Epub 2021 Feb 15. PMID: 33589826; PMCID: PMC8064272.
19. Chen, C., Mahjoubfar, A., Tai, LC. et al. Deep Learning in Label-free Cell Classification. Sci Rep 6, 21471 (2016). https://doi.org/10.1038/srep21471
20. Corin F. Otesteanu, Martina Ugrinic, Gregor Holzner, Yun-Tsan Chang, Christina Fassnacht, Emmanuella Guenova, Stavros Stavrakis, Andrew deMello, Manfred Claassen, A weakly supervised deep learning approach for label-free imaging flow-cytometry-based blood diagnostics, Cell Reports Methods, Volume 1, Issue 6, 2021, 100094, ISSN 2667-2375, https://doi.org/10.1016/j.crmeth.2021.100094.
21. Reddy OL, Stroncek DF, Panch SR. Improving CAR T cell therapy by optimizing critical quality attributes. Semin Hematol. 2020 Apr;57(2):33-38. doi: 10.1053/j.seminhematol.2020.07.005. Epub 2020 Jul 27. PMID: 32892841; PMCID: PMC7518470.
22. Hu Y, Huang J. The Chimeric Antigen Receptor Detection Toolkit. Front Immunol. 2020 Aug 11;11:1770. doi: 10.3389/fimmu.2020.01770. PMID: 32849635; PMCID: PMC7431616.
23. L. Tian, Z. Liu, L. Yeh *et al.*, “Computational illumination for high-speed in vitro Fourier ptychographic microscopy”, Optica, 2(10), 904-911 (2015)

24. Zheng G, Horstmeyer R, Yang C. Wide-field, high-resolution Fourier ptychographic microscopy. Nat Photonics. 2013 Sep 1;7(9):739-745. doi: 10.1038/nphoton.2013.187. PMID: 25243016; PMCID: PMC4169052.
25. Tian L, Waller L. Quantitative differential phase contrast imaging in an LED array microscope. Opt Express. 2015 May 4;23(9):11394-403. doi: 10.1364/OE.23.011394. PMID: 25969234.
26. Lei Tian, Jingyan Wang, and Laura Waller, "3D differential phase-contrast microscopy with computational illumination using an LED array," Opt. Lett. 39, 1326-1329 (2014)
27. Ziji Liu, Lei Tian, Sijia Liu, Laura Waller, "Real-time brightfield, darkfield, and phase contrast imaging in a light-emitting diode array microscope," J. Biomed. Opt. 19(10) 106002 (1 October 2014) https://doi.org/10.1117/1.JBO.19.10.106002
28. Pengzhi Li, Yan Pei, Jianqiang Li, A comprehensive survey on design and application of autoencoder in deep learning, Applied Soft Computing, Volume 138, 2023, 110176, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2023.110176.
29. Breiman, L. (2001) Random Forests. Machine Learning, 45, 5-32. https://doi.org/10.1023/A:1010933404324
30. Jin, X., Han, J. (2011). *K*-Means Clustering. In: Sammut, C., Webb, G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_425
31. Uddin S, Haque I, Lu H, Moni MA, Gide E. Comparative performance analysis of K-nearest neighbour (KNN) algorithm and its different variants for disease prediction. Sci Rep. 2022 Apr 15;12(1):6256. doi: 10.1038/s41598-022-10358-x. PMID: 35428863; PMCID: PMC9012855.

32. Alzubaidi, L., Zhang, J., Humaidi, A.J. *et al.* Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *J Big Data* 8, 53 (2021). https://doi.org/10.1186/s40537-021-00444-8

33. S. Chen and W. Guo, "Auto-Encoders in Deep Learning—A Review with New Perspectives," Mathematics 11, 1777 (2023).

34. Mousavikhamene, Z., Sykora, D.J., Mrksich, M. *et al.* Morphological features of single cells enable accurate automated classification of cancer from non-cancer cell lines. *Sci Rep* **11**, 24375 (2021). https://doi.org/10.1038/s41598-021-03813-8

35. Li, Y., Nowak, C.M., Pham, U. *et al.* Cell morphology-based machine learning models for human cell state classification. *npj Syst Biol Appl* **7**, 23 (2021). https://doi.org/10.1038/s41540-021-00180-y

36. Yao, K., Rochman, N.D. & Sun, S.X. Cell Type Classification and Unsupervised Morphological Phenotyping From Low-Resolution Images Using Deep Learning. *Sci Rep* **9**, 13467 (2019). https://doi.org/10.1038/s41598-019-50010-9

37. Dhar, P, Kothandapani, SD, Satti, SK, Padmanabhan, S. HPKNN: Hyper-parameter optimized KNN classifier for classification of poikilocytosis. *Int J Imaging Syst Technol*. 2023; 33(3): 928-950. doi:10.1002/ima.22841

38. Prakisya, Nurcahya Pradana Taufik, Liantoni, Febri, Hatta, Puspanda, Aristyagama, Yusfia Hafid and Setiawan, Andika. "Utilization of *K*-nearest neighbor algorithm for classification of white blood cells in AML M4, M5, and M7" *Open Engineering*, vol. 11, no. 1, 2021, pp. 662-668. https://doi.org/10.1515/eng-2021-0065

39. Marklein, R.A., Lo Surdo, J.L., Bellayr, I.H., Godil, S.A., Puri, R.K. and Bauer, S.R. (2016), High Content Imaging of Early Morphological Signatures Predicts Long Term

Mineralization Capacity of Human Mesenchymal Stem Cells upon Osteogenic Induction. Stem Cells, 34: 935-947. https://doi.org/10.1002/stem.2322

40. Kanupriya R. Daga, Priyanka Priyadarshani, Andrew M. Larey, Kejie Rui, Luke J. Mortensen, Ross A. Marklein, Shape up before you ship out: morphology as a potential critical quality attribute for cellular therapies, Current Opinion in Biomedical Engineering, Volume 20, 2021, 100352, ISSN 2468-4511, https://doi.org/10.1016/j.cobme.2021.100352.
41. Walsh, A.J., Mueller, K.P., Tweed, K. *et al.* Classification of T-cell activation via autofluorescence lifetime imaging. *Nat Biomed Eng* **5**, 77–88 (2021). https://doi.org/10.1038/s41551-020-0592-z
42. Danielle E. Desa, Tongcheng Qian, Melissa C. Skala, Label-free optical imaging and sensing for quality control of stem cell manufacturing, Current Opinion in Biomedical Engineering, Volume 25, 2023, 100435, ISSN 2468-4511, https://doi.org/10.1016/j.cobme.2022.100435.
43. Rupsa Datta, Tiffany M. Heaster, Joe T. Sharick, Amani A. Gillette, Melissa C. Skala, "Fluorescence lifetime imaging microscopy: fundamentals and advances in instrumentation, analysis, and applications," J. Biomed. Opt. 25(7) 071203 (13 May 2020) https://doi.org/10.1117/1.JBO.25.7.071203
44. Kayvan Samimi, Emmanuel Contreras Guzman, Steven M. Trier, Dan L. Pham, Tongcheng Qian, and Melissa C. Skala, "Time-domain single photon-excited autofluorescence lifetime for label-free detection of T cell activation," Opt. Lett. 46, 2168-2171 (2021)

45. K. P. Mueller et al., “Production and characterization of virus-free, CRISPR-CAR T cells capable of inducing solid tumor regression,” J Immunother Cancer, vol. 10, no. 9, 2022, doi: 10.1136/jitc-2021-004446.

APPENDIX A

CELLPROFILER MEASUREMENTS

General List of Measurements taken from CellProfiler, where CART is the name of the image in CellProfiler, and cells is the name of the final object being measured. Information on these metrics can be found in the documentation for the program.

- AreaShape_Area
- AreaShape_BoundingBoxArea
- AreaShape_BoundingBoxMaximum_X
- AreaShape_BoundingBoxMaximum_Y
- AreaShape_BoundingBoxMinimum_X
- AreaShape_BoundingBoxMinimum_Y
- AreaShape_Center_X
- AreaShape_Center_Y
- AreaShape_Compactness
- AreaShape_ConvexArea
- AreaShape_Eccentricity
- AreaShape_EquivalentDiameter
- AreaShape_EulerNumber
- AreaShape_Extent
- AreaShape_FormFactor
- AreaShape_MajorAxisLength
- AreaShape_MaxFeretDiameter
- AreaShape_MaximumRadius
- AreaShape_MeanRadius
- AreaShape_MedianRadius
- AreaShape_MinFeretDiameter
- AreaShape_MinorAxisLength
- AreaShape_Orientation
- AreaShape_Perimeter
- AreaShape_Solidity

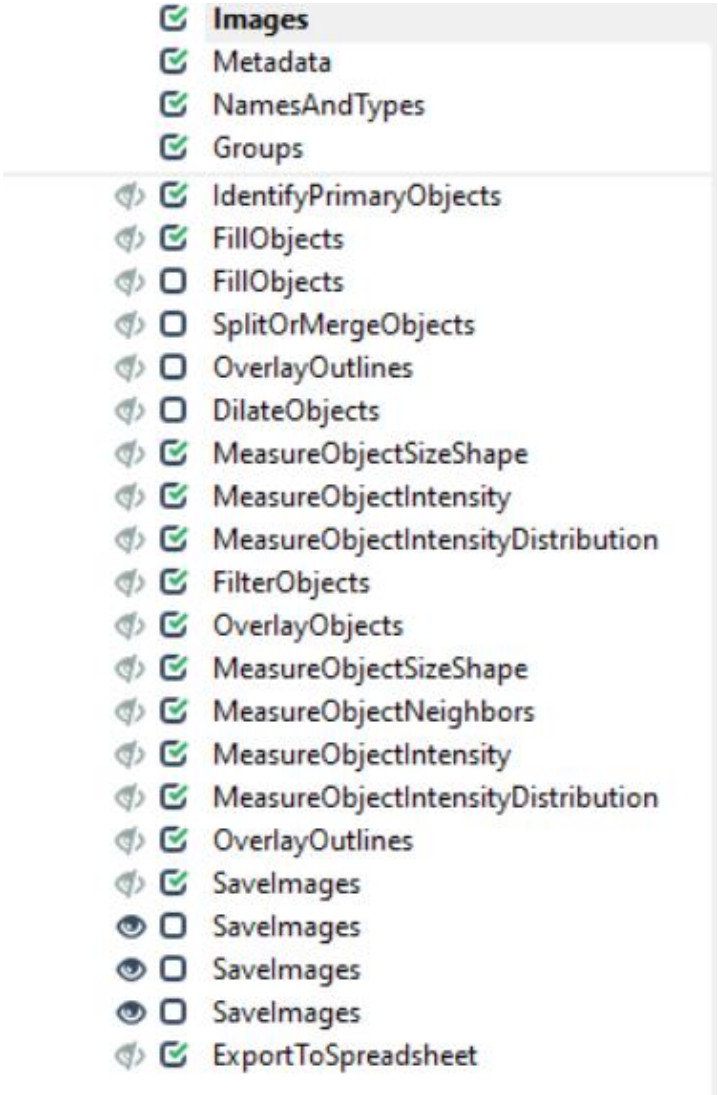
- Intensity_IntegratedIntensityEdge_CART
- Intensity_IntegratedIntensity_CART
- Intensity_LowerQuartileIntensity_CART
- Intensity_MADIntensity_CART
- Intensity_MassDisplacement_CART
- Intensity_MaxIntensityEdge_CART
- Intensity_MaxIntensity_CART
- Intensity_MeanIntensityEdge_CART
- Intensity_MeanIntensity_CART
- Intensity_MedianIntensity_CART
- Intensity_MinIntensityEdge_CART
- Intensity_MinIntensity_CART
- Intensity_StdIntensityEdge_CART
- Intensity_StdIntensity_CART
- Intensity_UpperQuartileIntensity_CART
- Location_CenterMassIntensity_X_CART
- Location_CenterMassIntensity_Y_CART
- Location_CenterMassIntensity_Z_CART
- Location_Center_X
- Location_Center_Y
- Location_Center_Z
- Location_MaxIntensity_X_CART
- Location_MaxIntensity_Y_CART
- Location_MaxIntensity_Z_CART
- Neighbors_AngleBetweenNeighbors_3
- Neighbors_FirstClosestDistance_3
- Neighbors_FirstClosestObjectNumber_3
- Neighbors_NumberOfNeighbors_3
- Neighbors_PercentTouching_3
- Neighbors_SecondClosestDistance_3
- Neighbors_SecondClosestObjectNumber_3
- Number_Object_Number
- RadialDistribution_FracAtD_CART_1of4
- RadialDistribution_FracAtD_CART_2of4
- RadialDistribution_FracAtD_CART_3of4
- RadialDistribution_FracAtD_CART_4of4
- RadialDistribution_MeanFrac_CART_1of4
- RadialDistribution_MeanFrac_CART_2of4
- RadialDistribution_MeanFrac_CART_3of4
- RadialDistribution_MeanFrac_CART_4of4
- RadialDistribution_RadialCV_CART_1of4

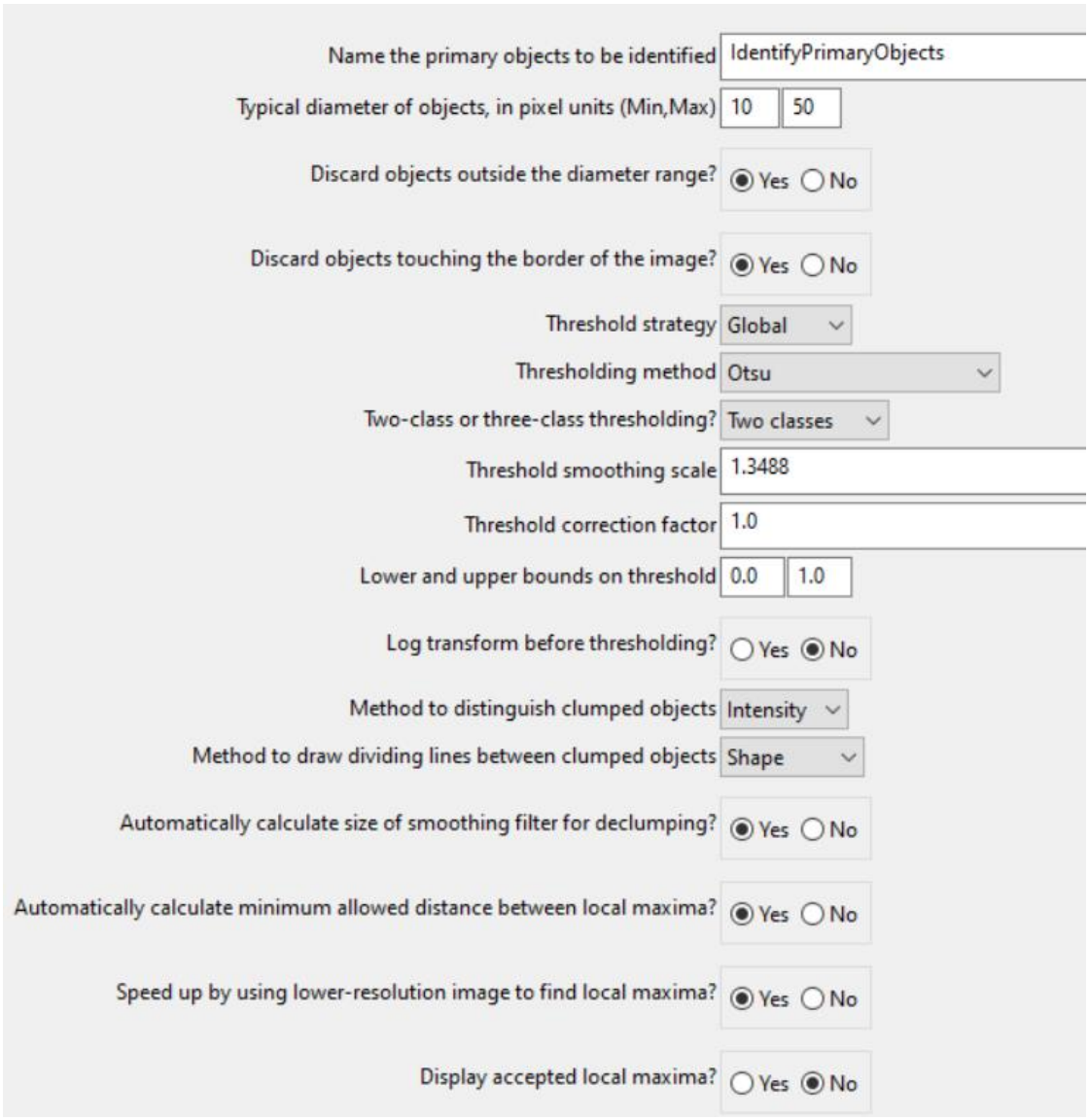
- RadialDistribution_RadialCV_CART_2of4
- RadialDistribution_RadialCV_CART_3of4
- RadialDistribution_RadialCV_CART_4of4
- RadialDistribution_ZernikeMagnitude_CART_0_0 through 9_9
- RadialDistribution_ZernikePhase_CART_0_0 through 9_9
- AreaShape_Zernike_0_0 through 9_9

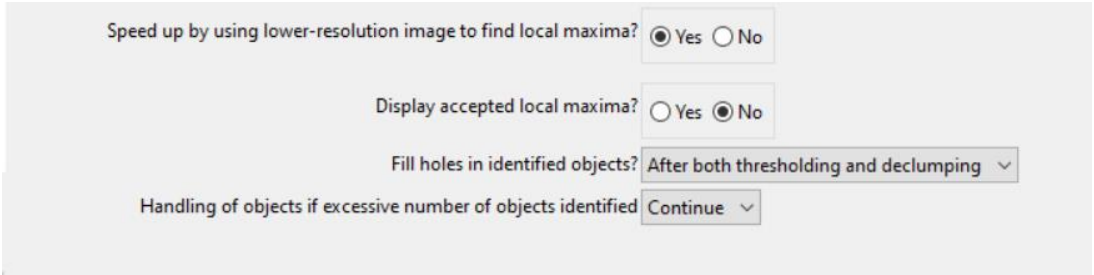
APPENDIX B

CELLPROFILER PIPELINE EXAMPLE

Example of pipeline and initial segmentation settings.




Name the primary objects to be identified
IdentifyPrimaryObjects
Typical diameter of objects, in pixel units (Min,Max)
10
50
Discard objects outside the diameter range?
Yes
No
Discard objects touching the border of the image?
Yes
No
Threshold strategy
Global
Thresholding method
Otsu
Two-class or three-class thresholding?
Two classes
Threshold smoothing scale
1.3488
Threshold correction factor
1.0
Lower and upper bounds on threshold
0.0
1.0
Log transform before thresholding?
Yes
No
Method to distinguish clumped objects
Intensity
Method to draw dividing lines between clumped objects
Shape
Automatically calculate size of smoothing filter for declumping?
Yes
No
Automatically calculate minimum allowed distance between local maxima?
Yes
No
Speed up by using lower-resolution image to find local maxima?
Yes
No
Display accepted local maxima?
Yes
No

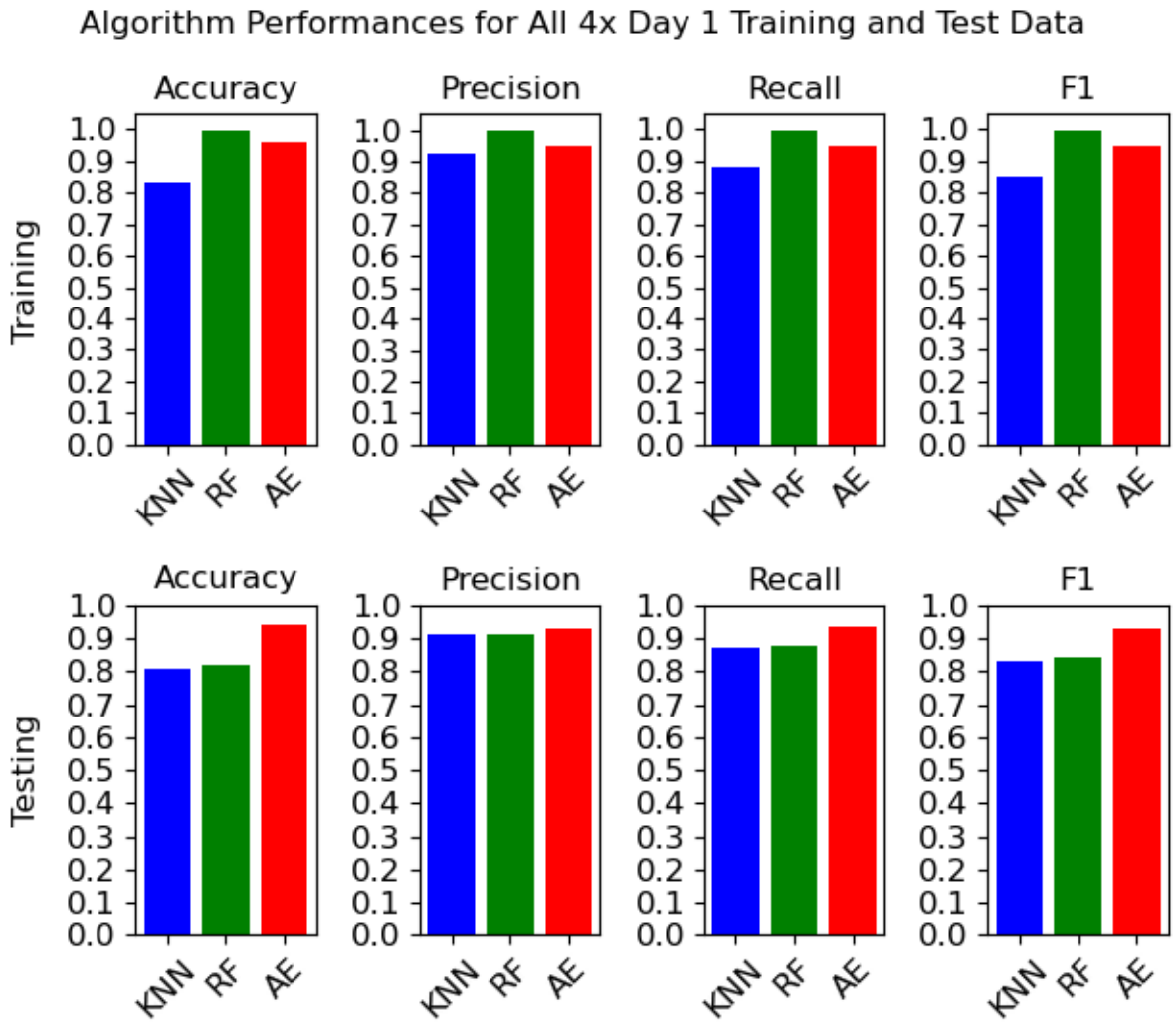

Speed up by using lower-resolution image to find local maxima?
Yes
No
Display accepted local maxima?
Yes
No
Fill holes in identified objects?
After both thresholding and declumping
Handling of objects if excessive number of objects identified
Continue

APPENDIX C

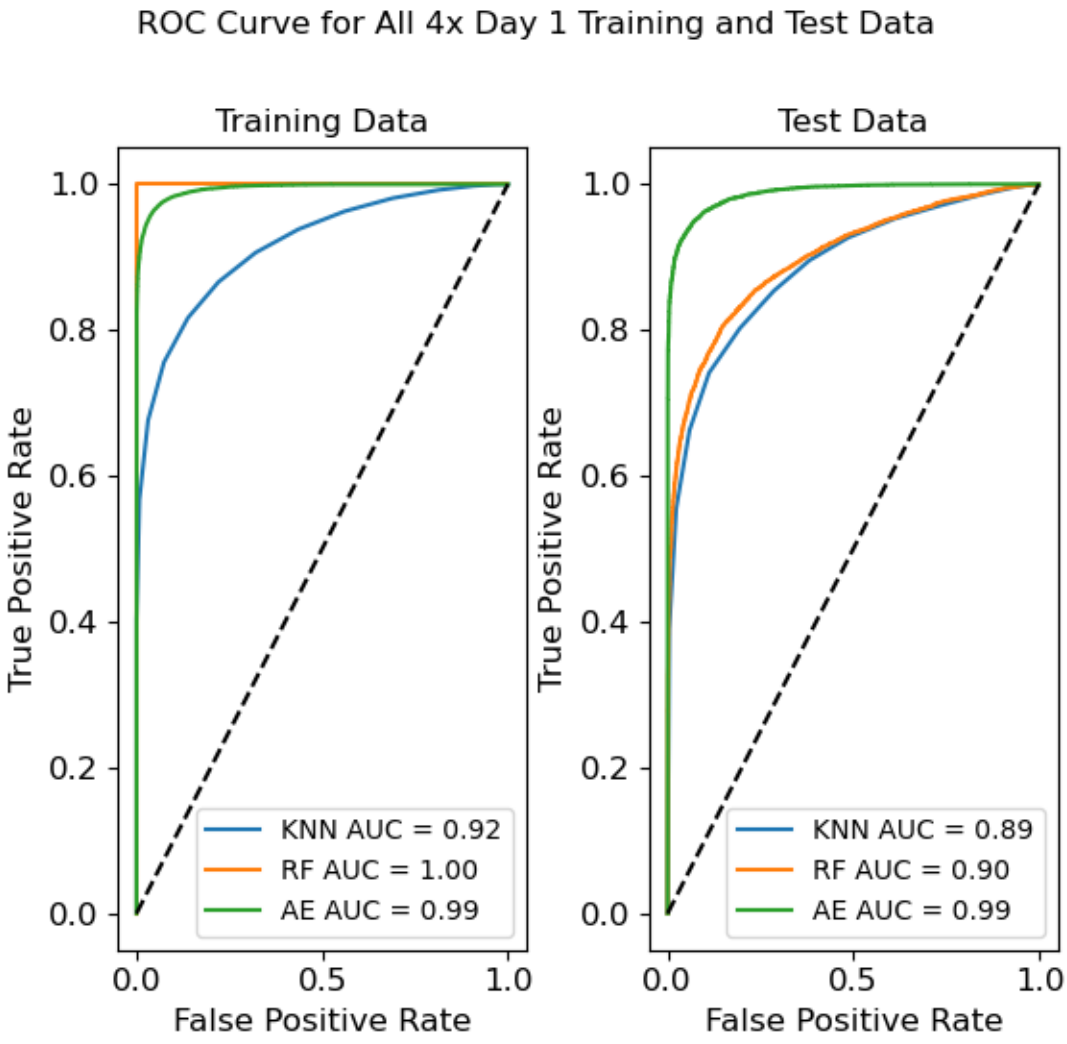
ADDITIONAL MACHINE LEARNING DATA

Aggregate 4x Data Figures

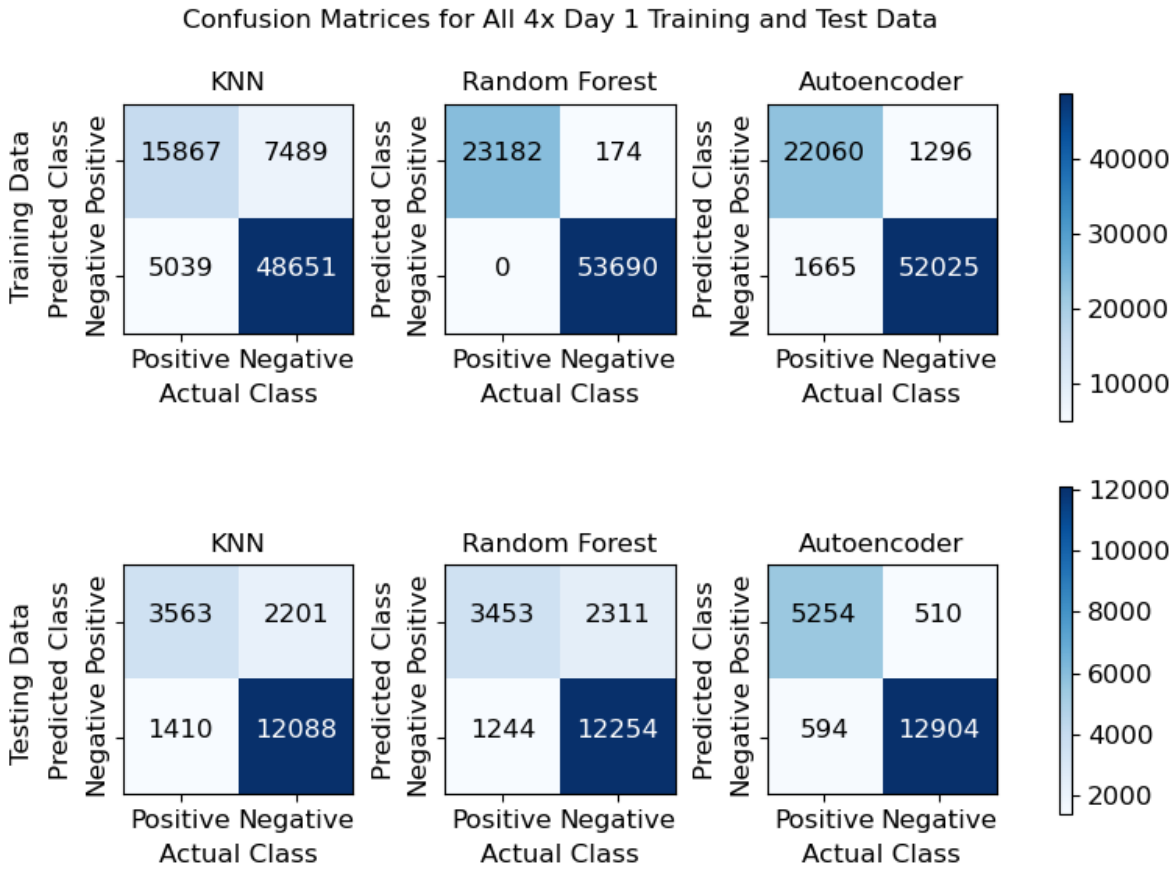
Aggregate Data Evaluation 4x Day 1



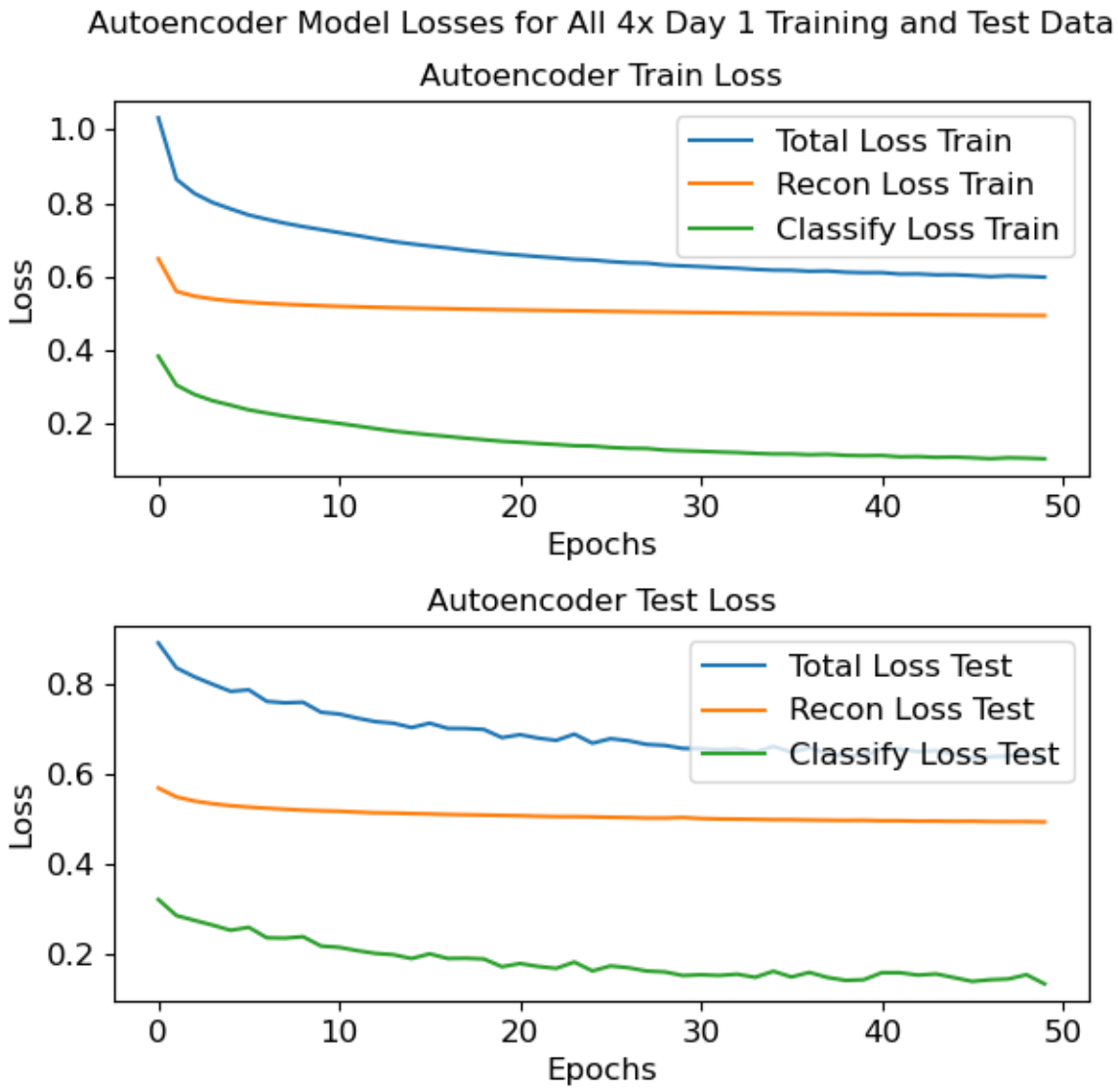
Algorithm Performances for All 4x Day 1 Training and Test Data



ROC Curve for All 4x Day 1 Training and Test Data

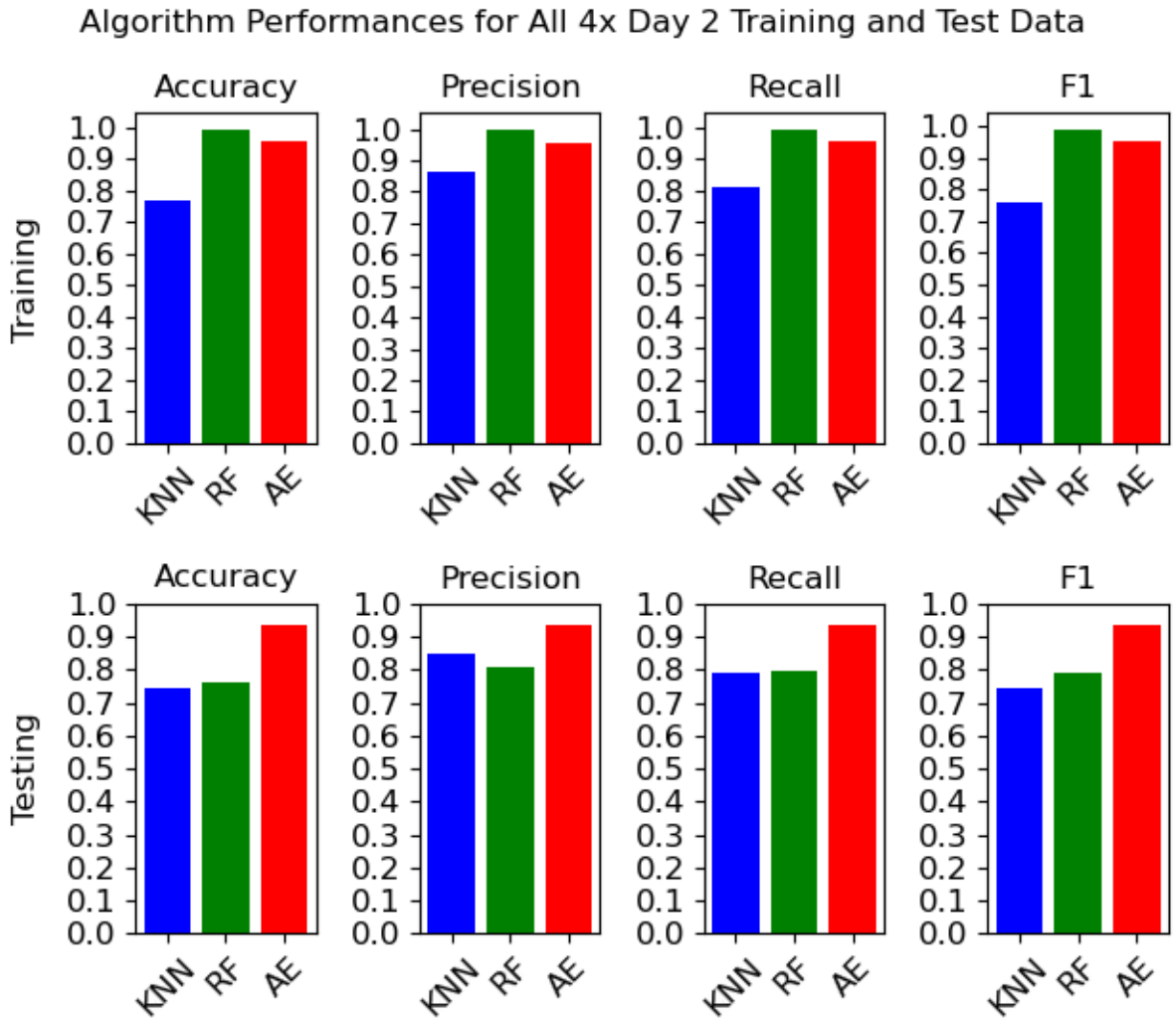


Confusion Matrices for All 4x Day 1 Training and Test Data

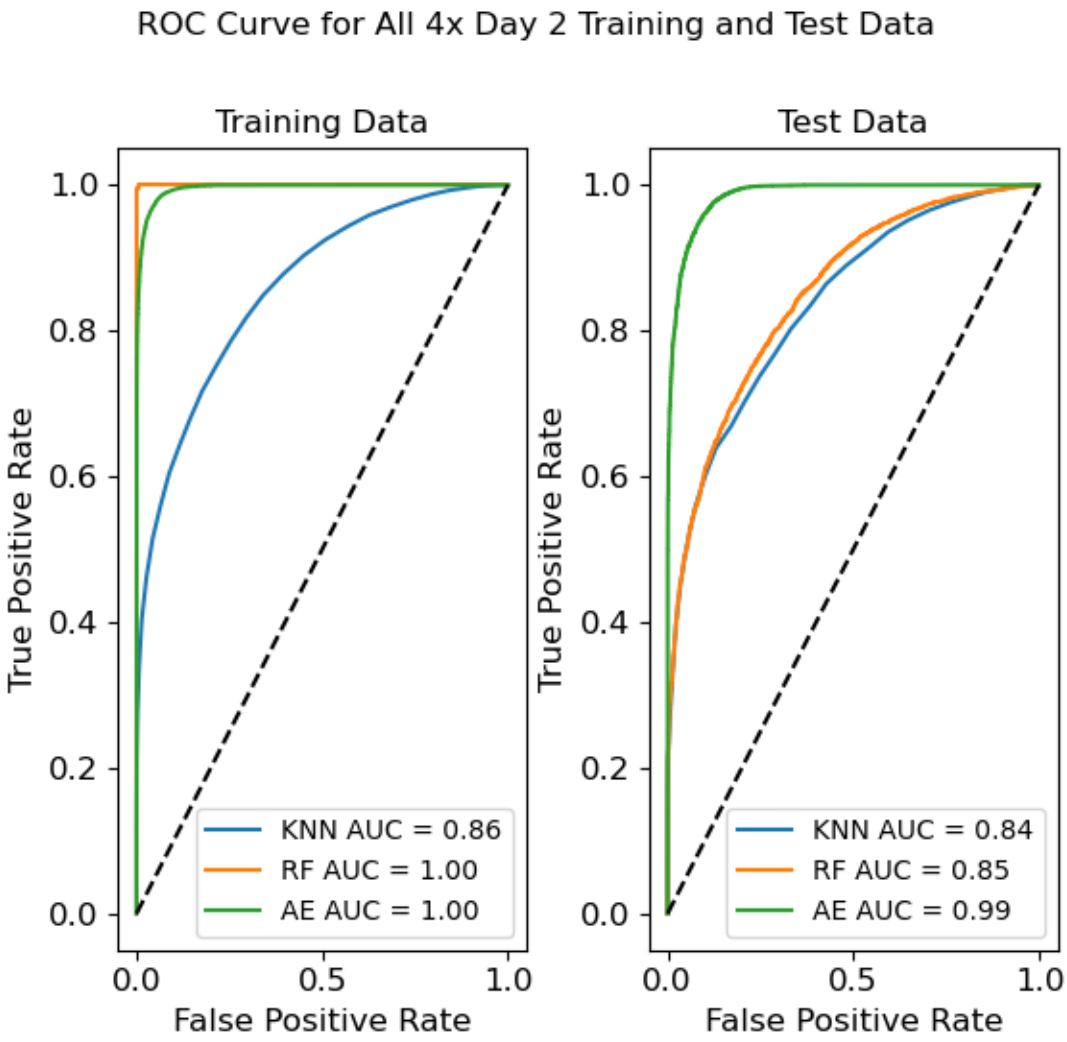


Autoencoder Model Losses for All 4x Day 1 Training and Test Data

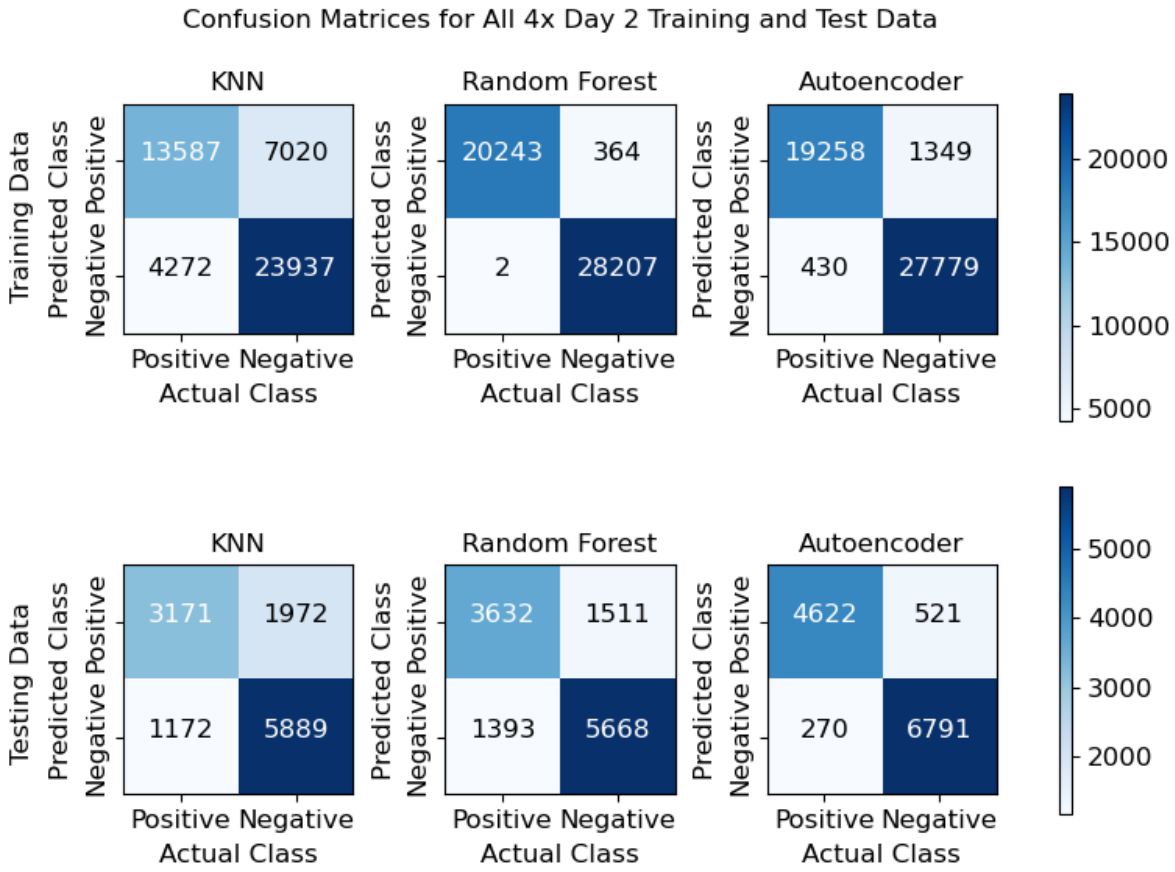
Aggregate Data Evaluation 4x Day 2



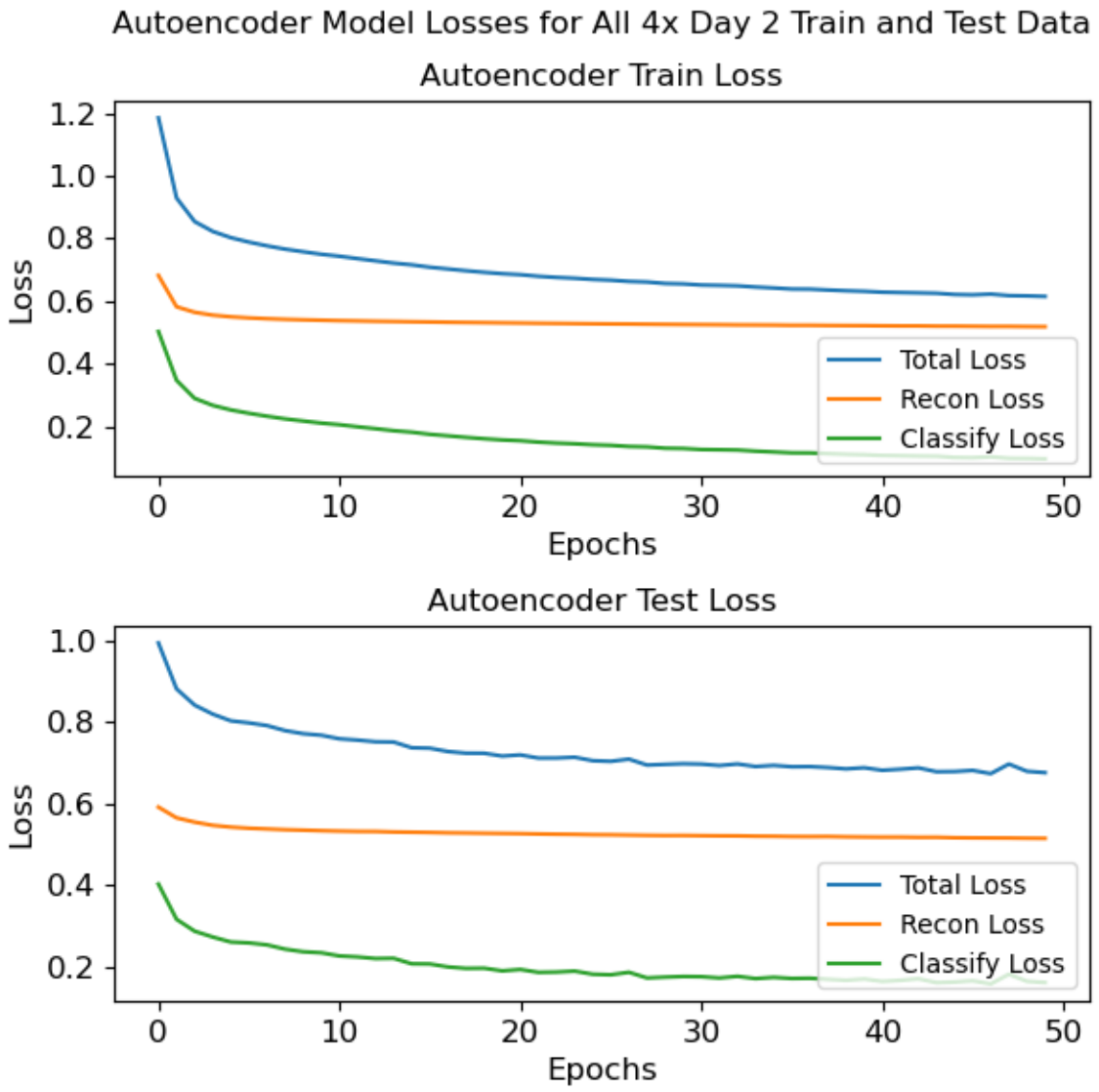
Algorithm Performances for All 4x Day 2 Training and Test Data



ROC Curve for All 4x Day 2 Training and Test Data

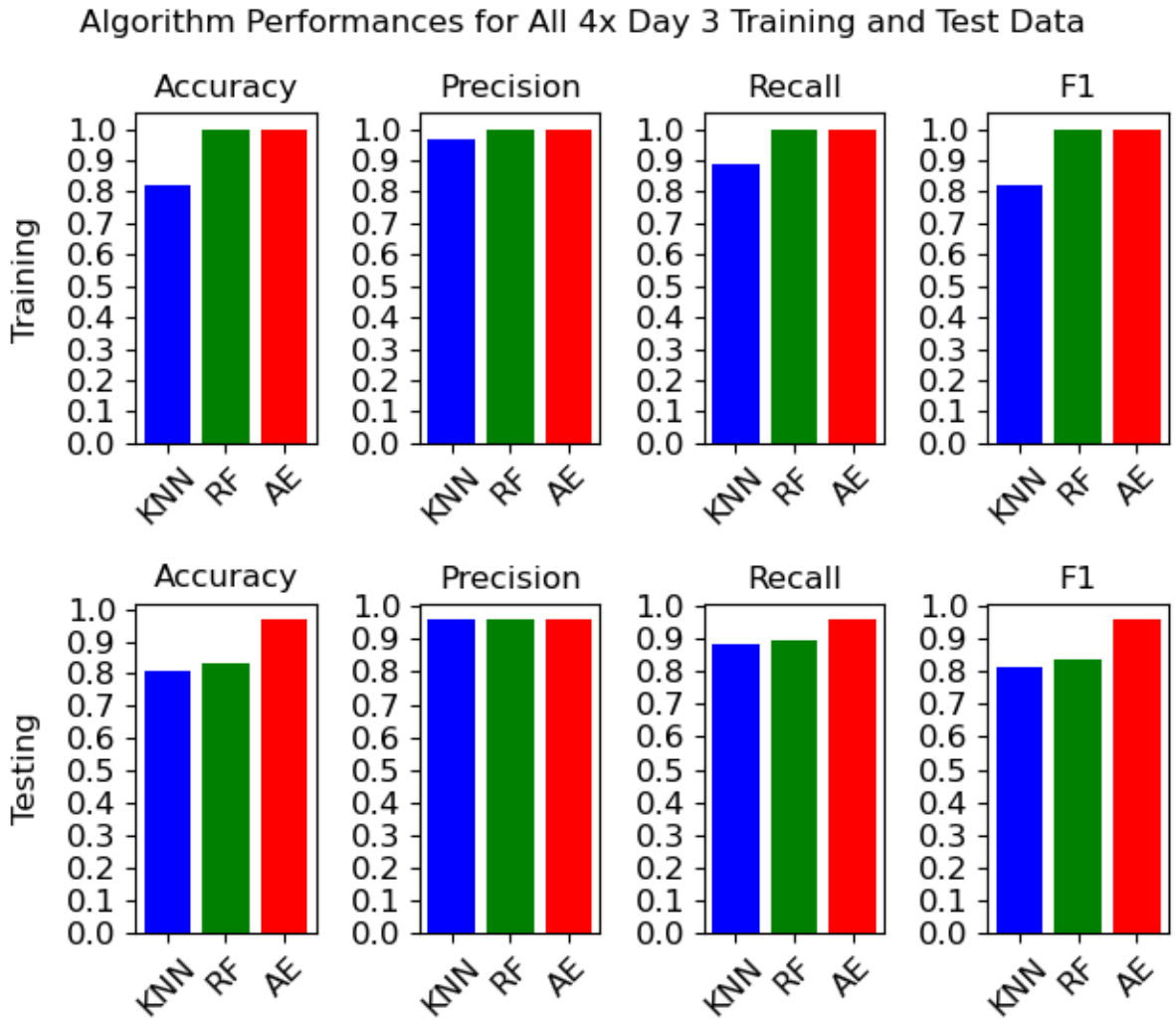


Confusion Matrices for All 4x Day 2 Training and Test Data

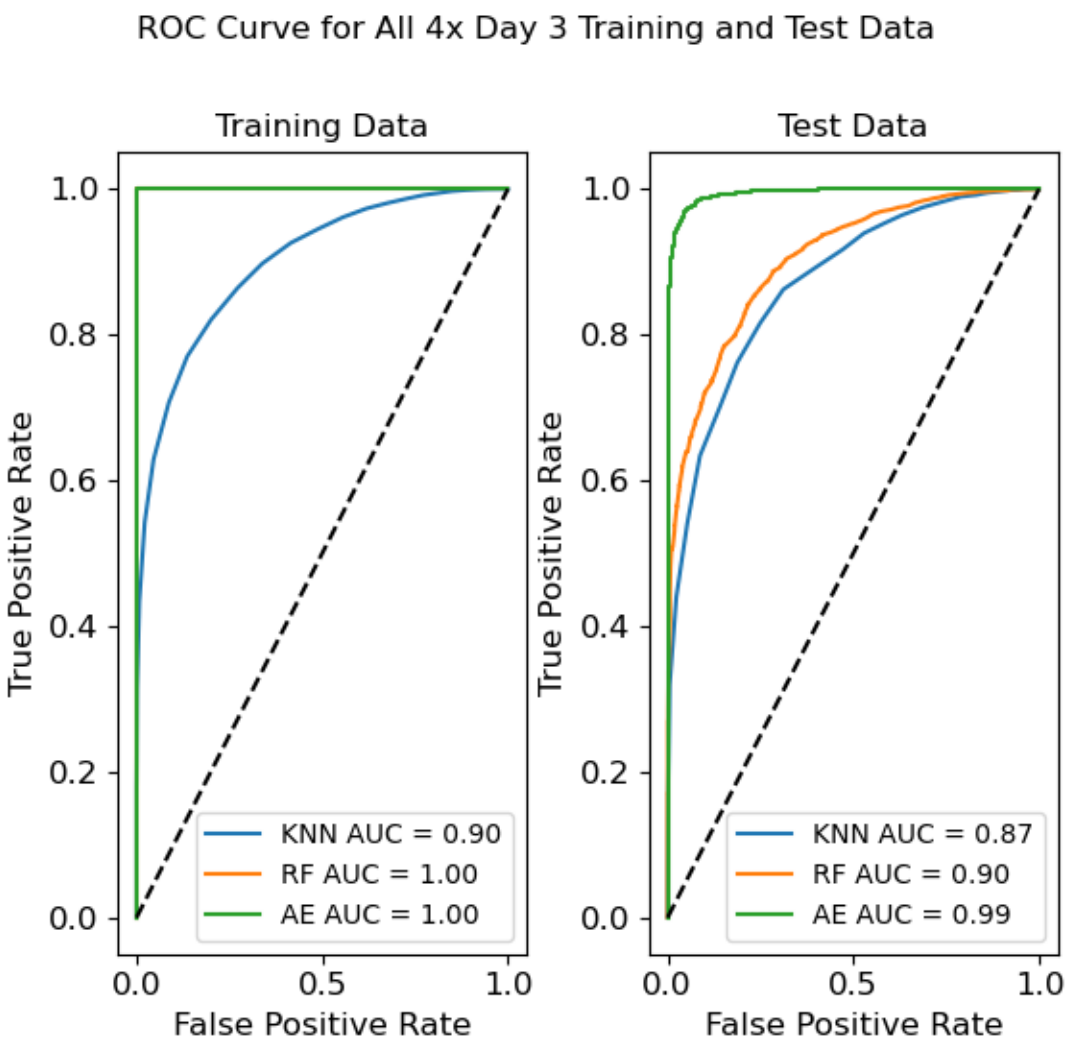


Autoencoder Model Losses for All 4x Day 2 Train and Test Data

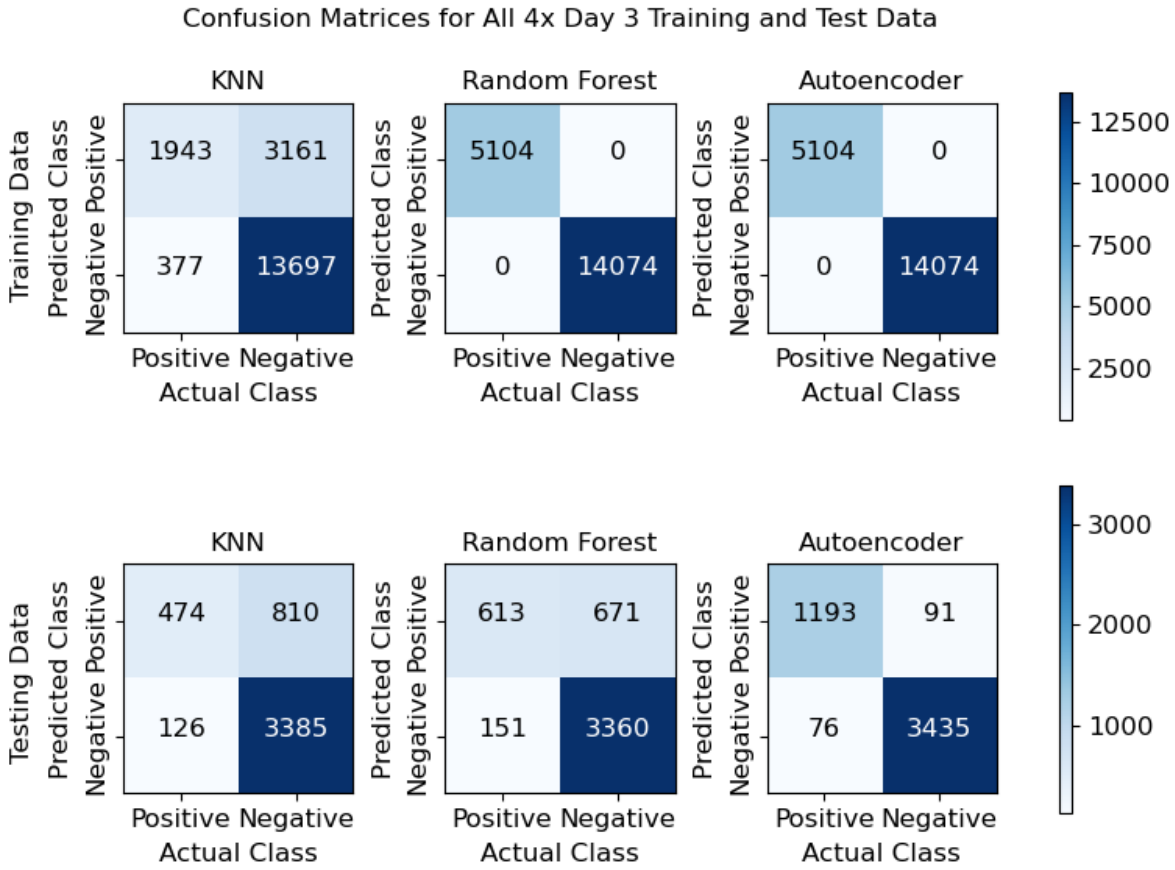
Aggregate Data Evaluation 4x Day 3



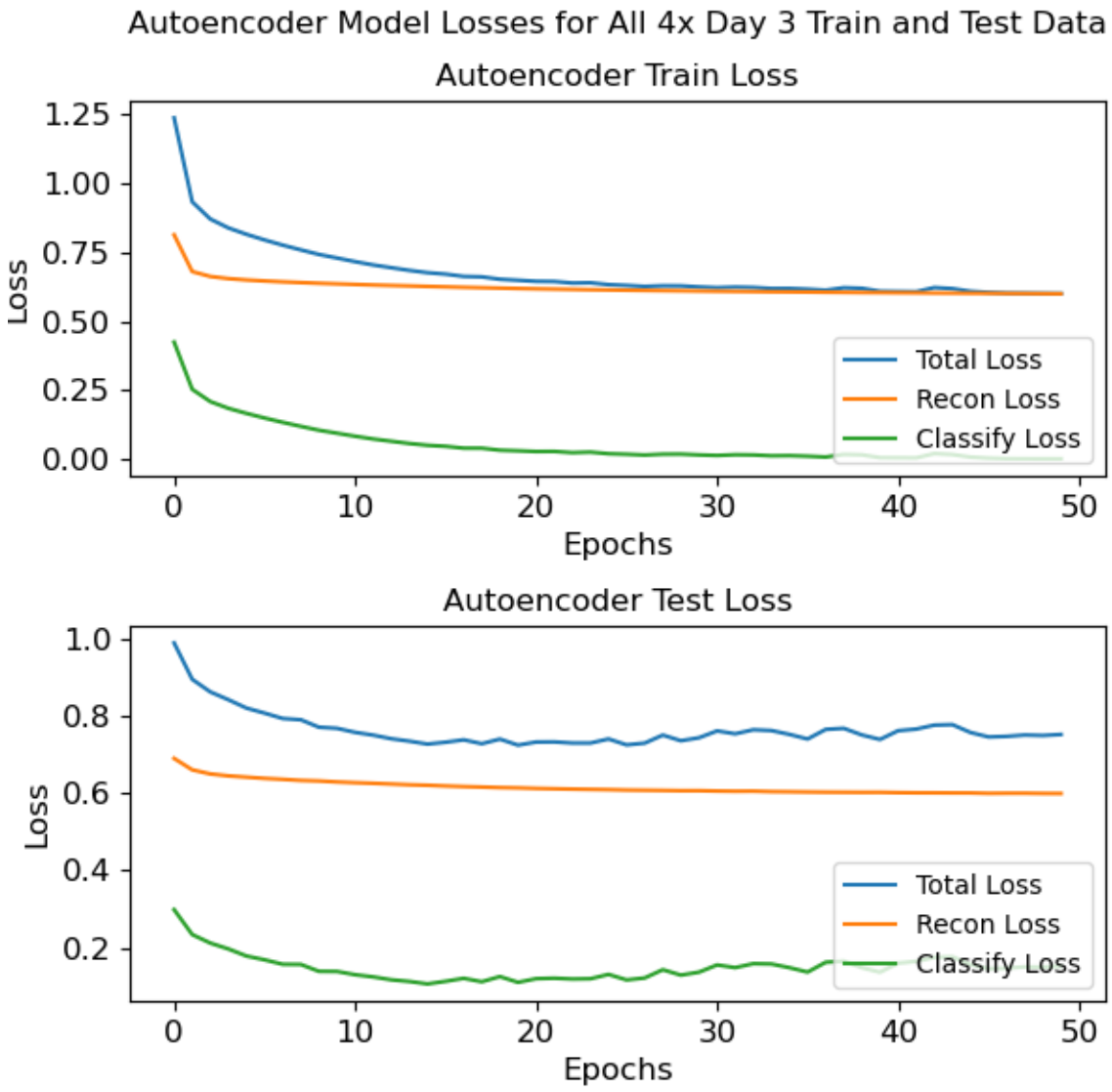
Algorithm Performances for All 4x Day 3 Training and Test Data



ROC Curve for All 4x Day 3 Training and Test Data



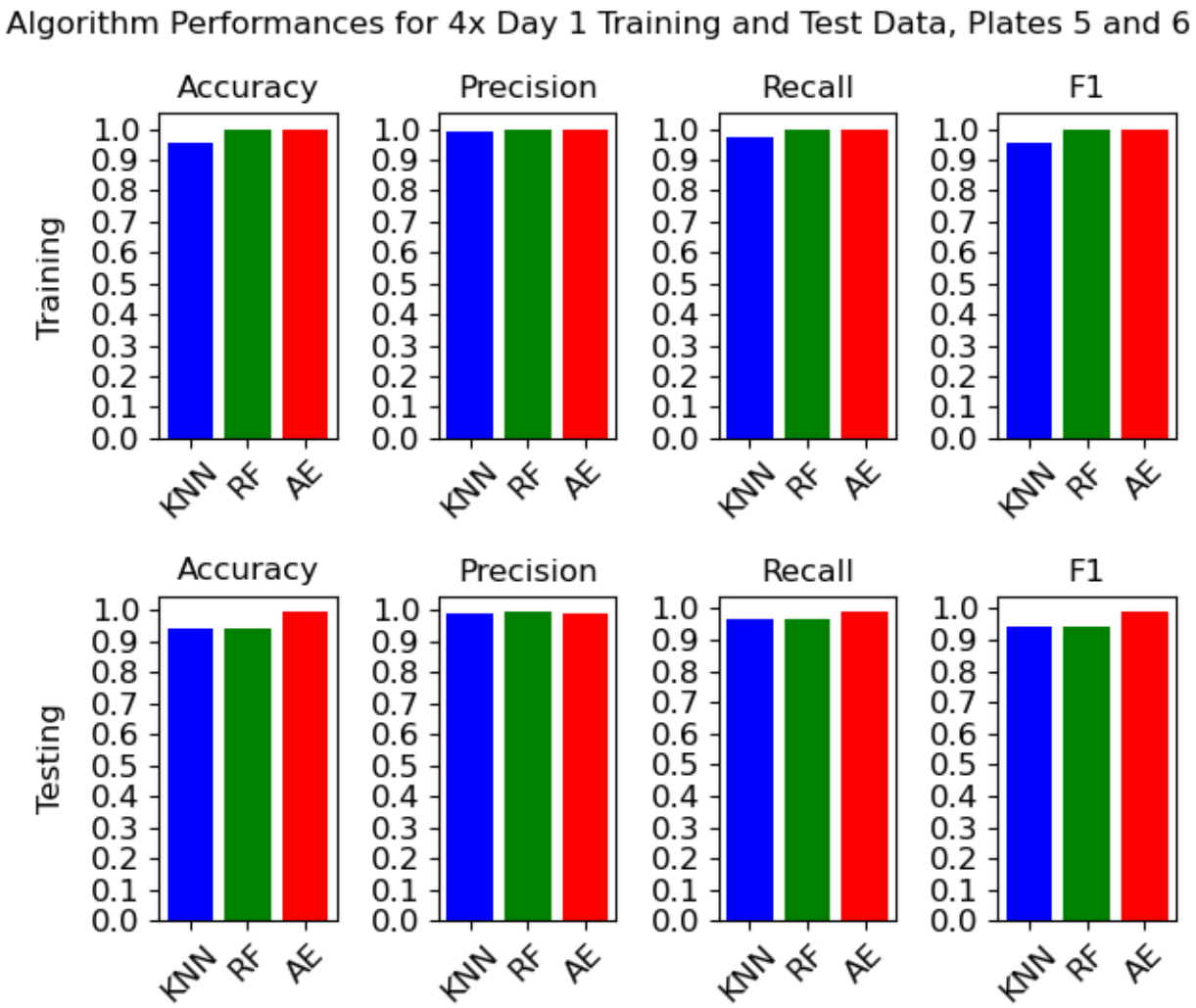
Confusion Matrices for All 4x Day 3 Training and Test Data



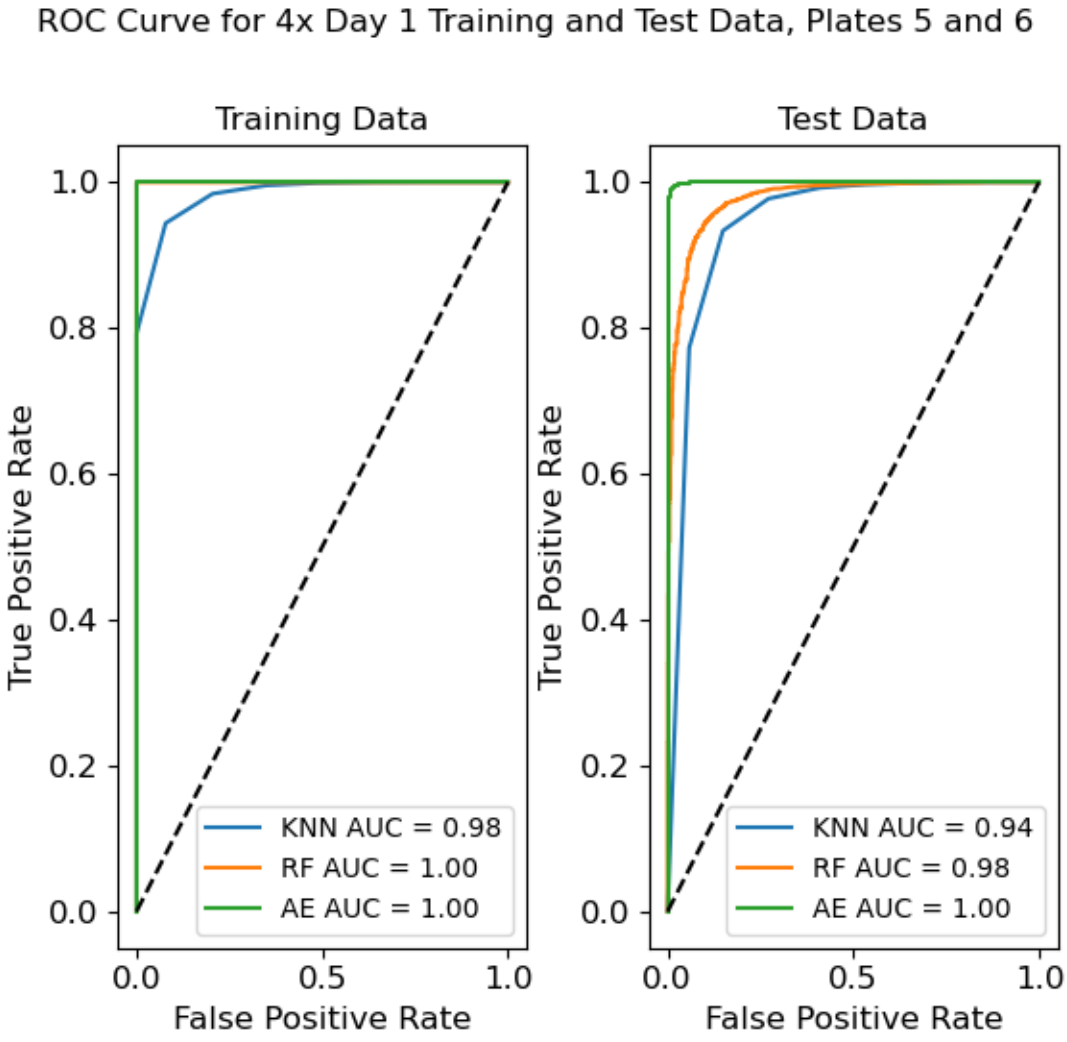
Autoencoder Model Losses for All 4x Day 3 Train and Test Data

Plates 5 and 6 4x Data Figures

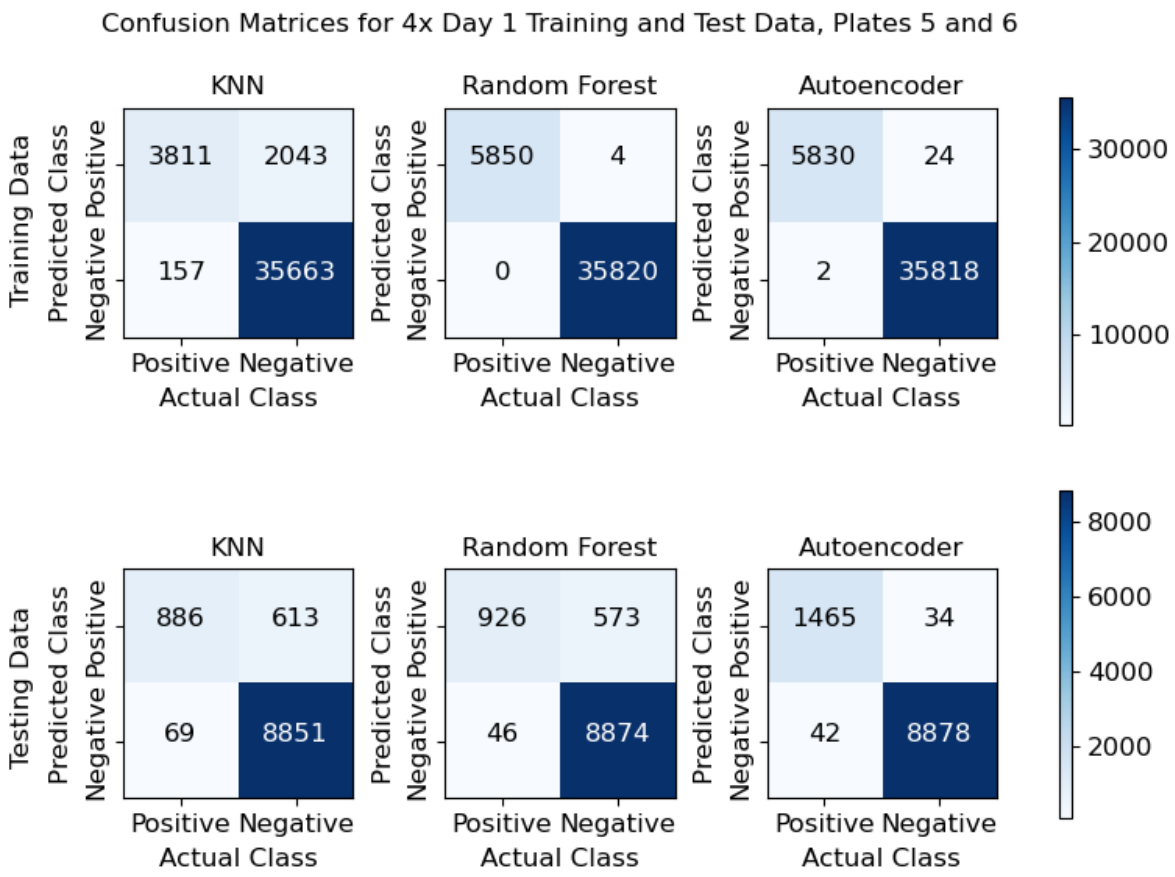
Plates 5 and 6 Day 1



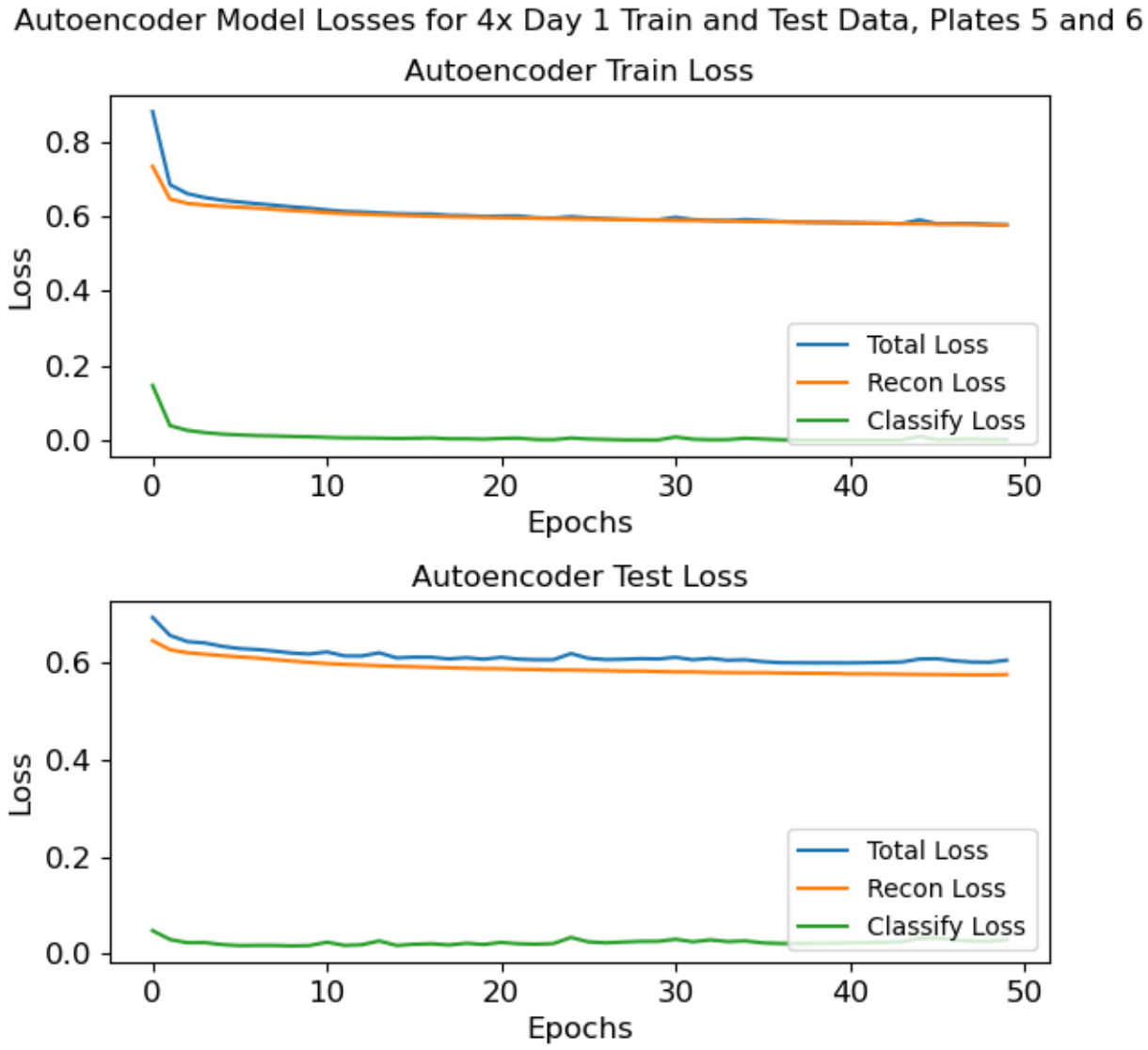
Algorithm Performances for 4x Day 1 Training and Test Data, Plates 5 and 6



ROC Curve for 4x for Day 1 Training and Test Data, Plates 5 and 6

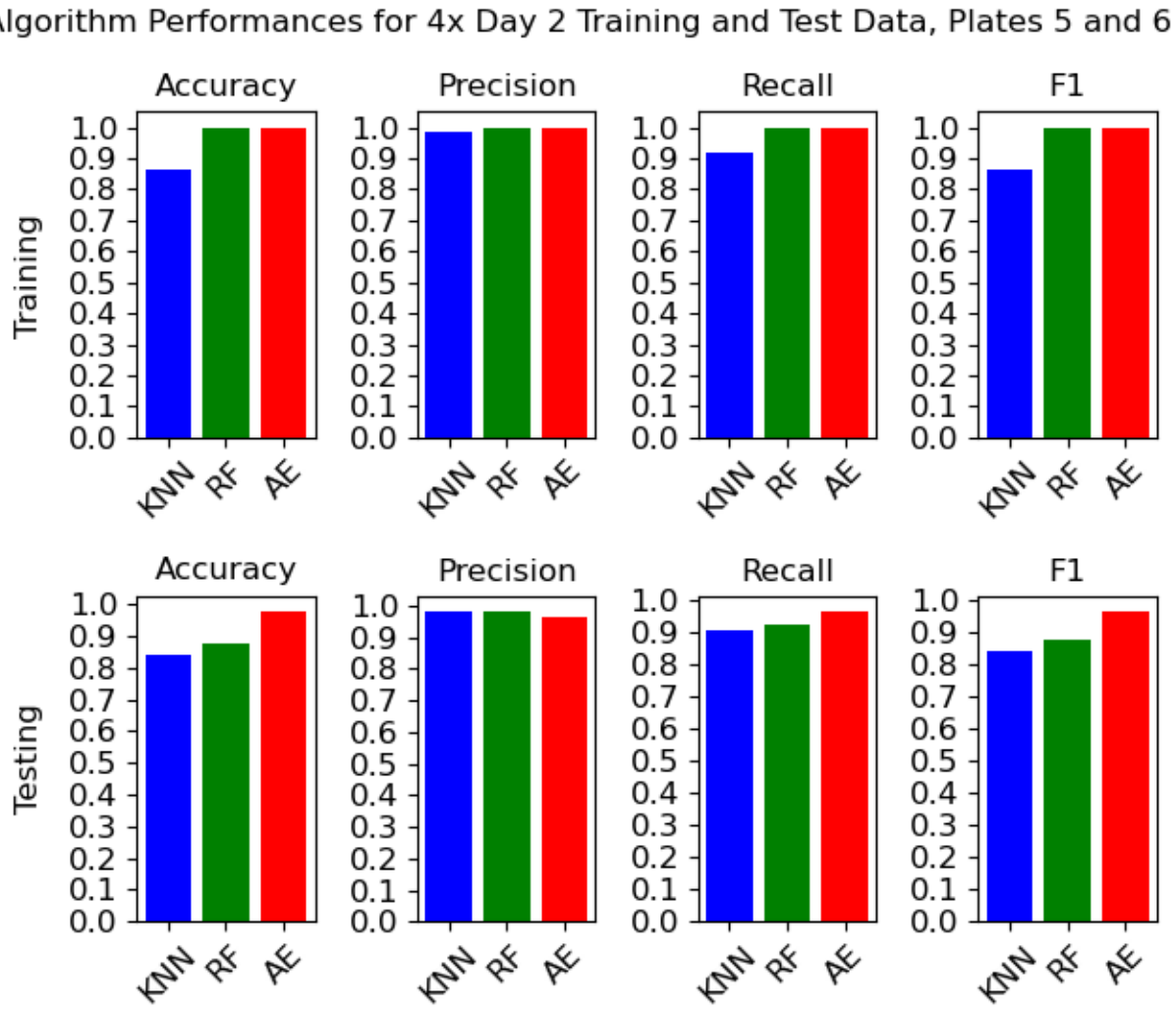


Confusion Matrices for 4x Day 1 Training and Test Data, Plates 5 and 6

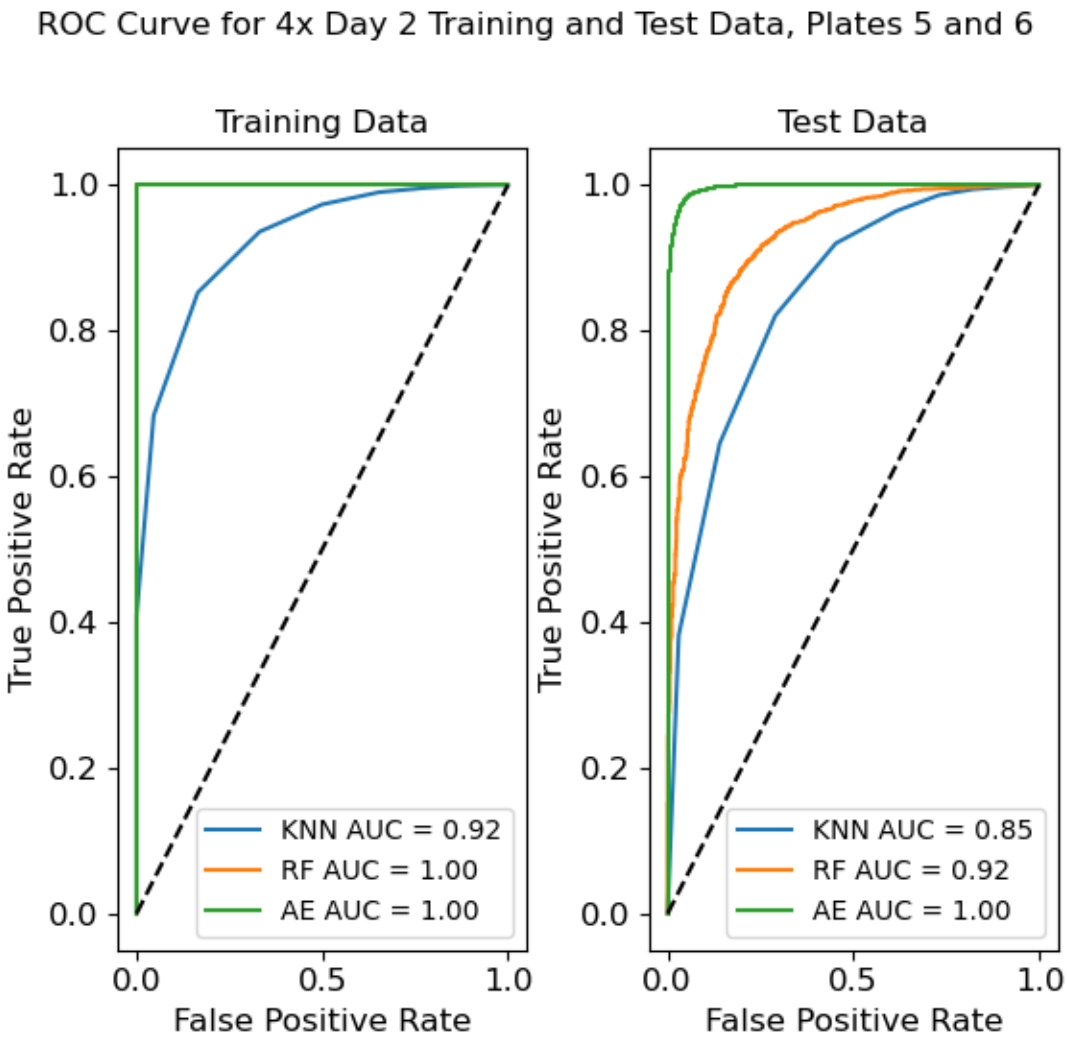


Autoencoder Model Losses for 4x Day 1 Train and Test Data, Plates 5 and 6

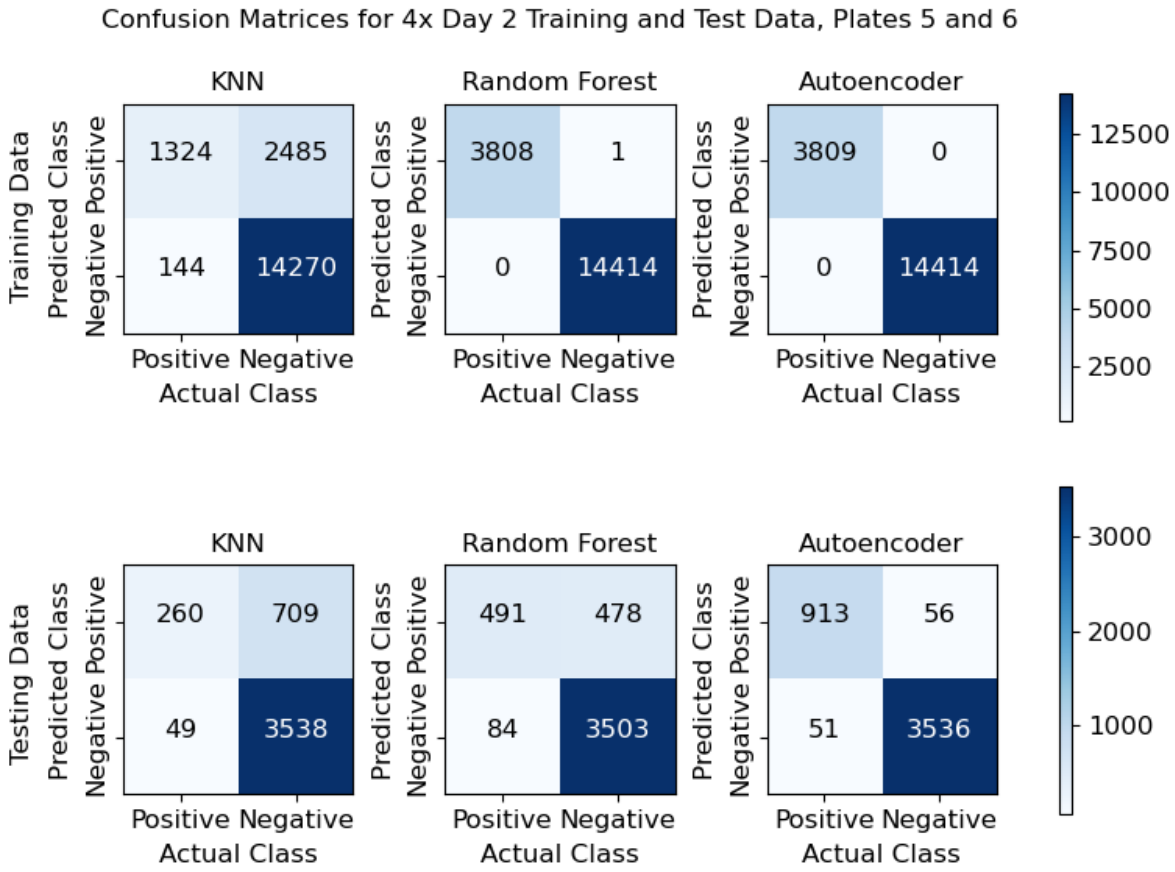
Plates 5 and 6 Day 2



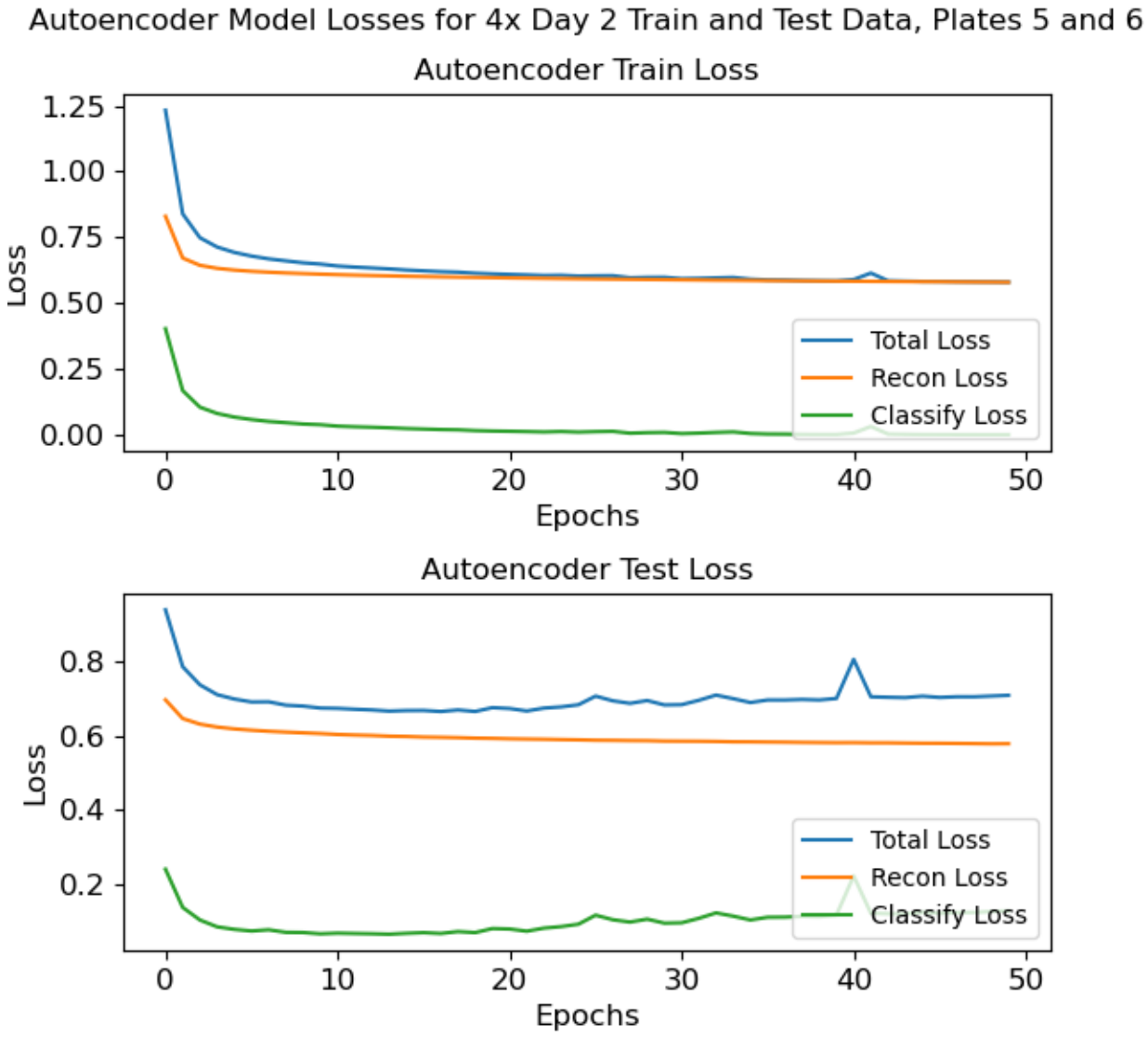
Algorithm Performances for 4x Day 2 Training and Test Data, Plates 5 and 6



ROC Curve for 4x for Day 2 Training and Test Data, Plates 5 and 6

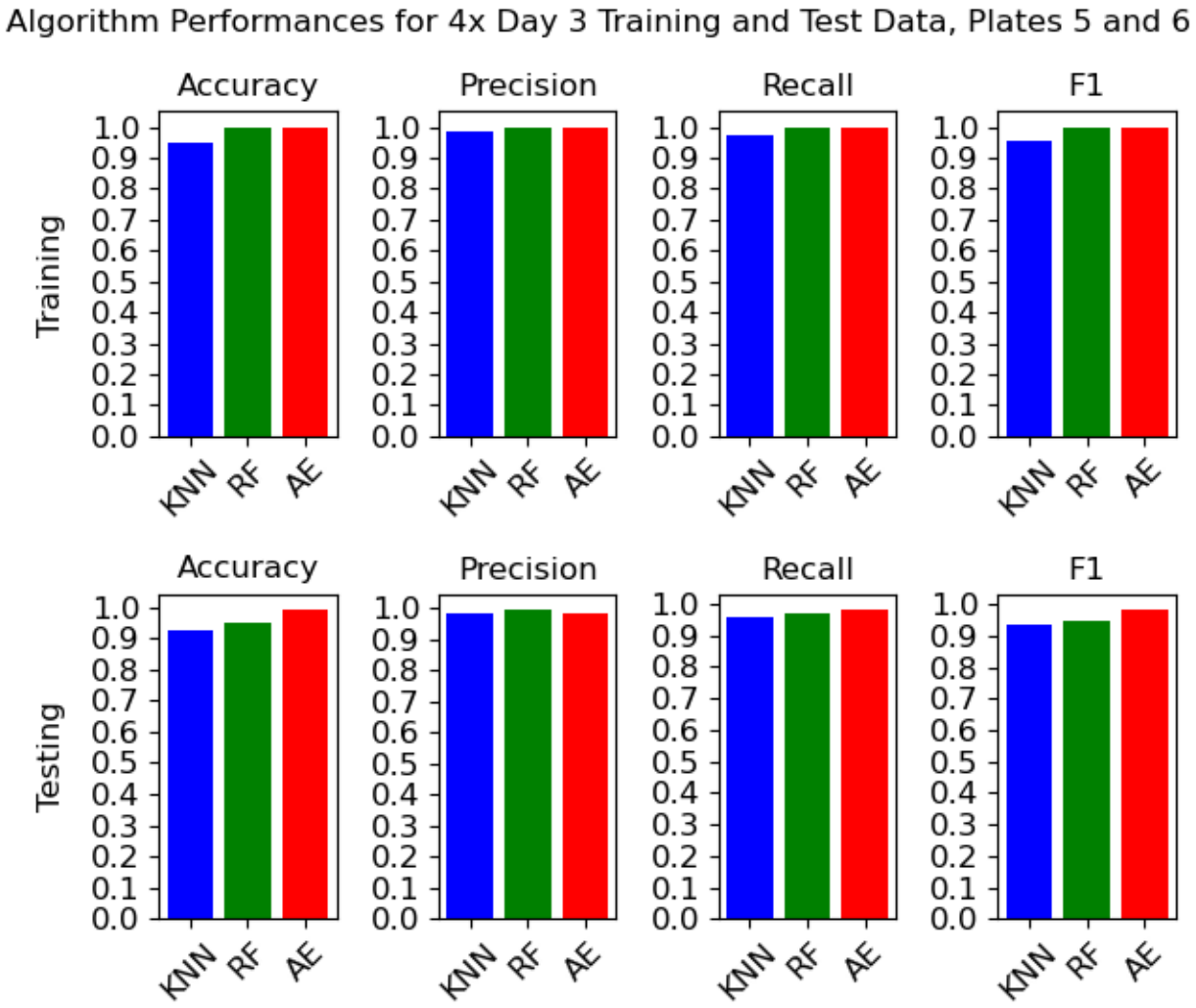


Autoencoder Classification for 4x Day 2 Train and Test Data, Plates 5 and 6

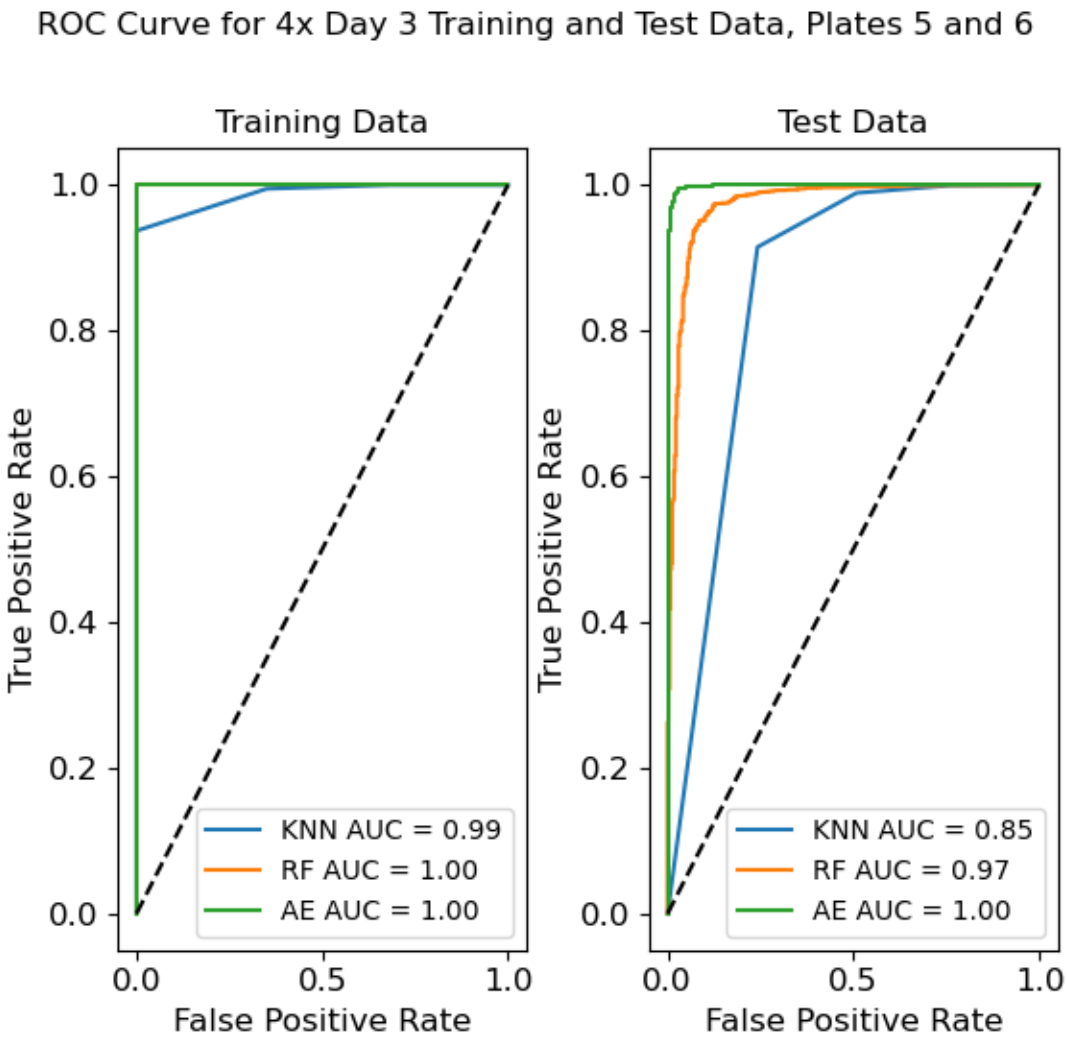


Autoencoder Model Losses for 4x Day 2 Train and Test Data, Plates 5 and 6

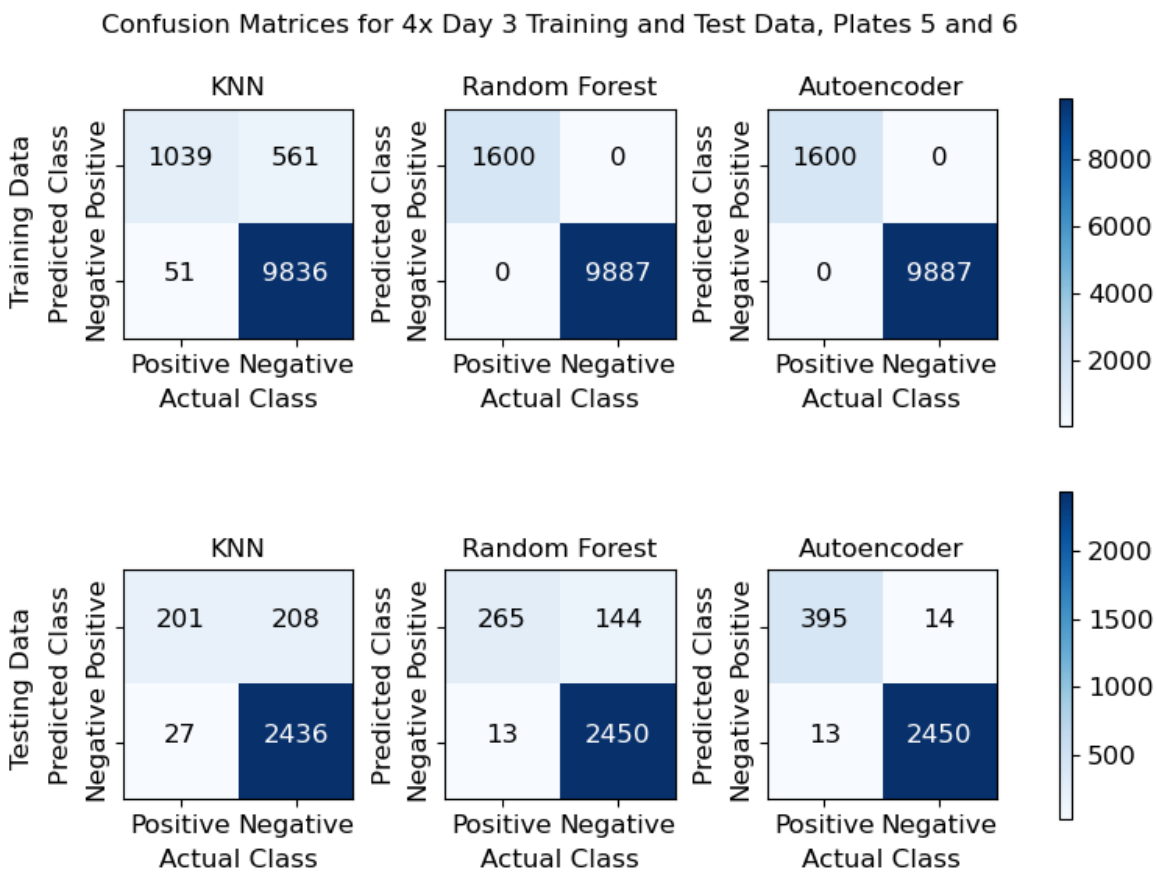
Plates 5 and 6 Day 3



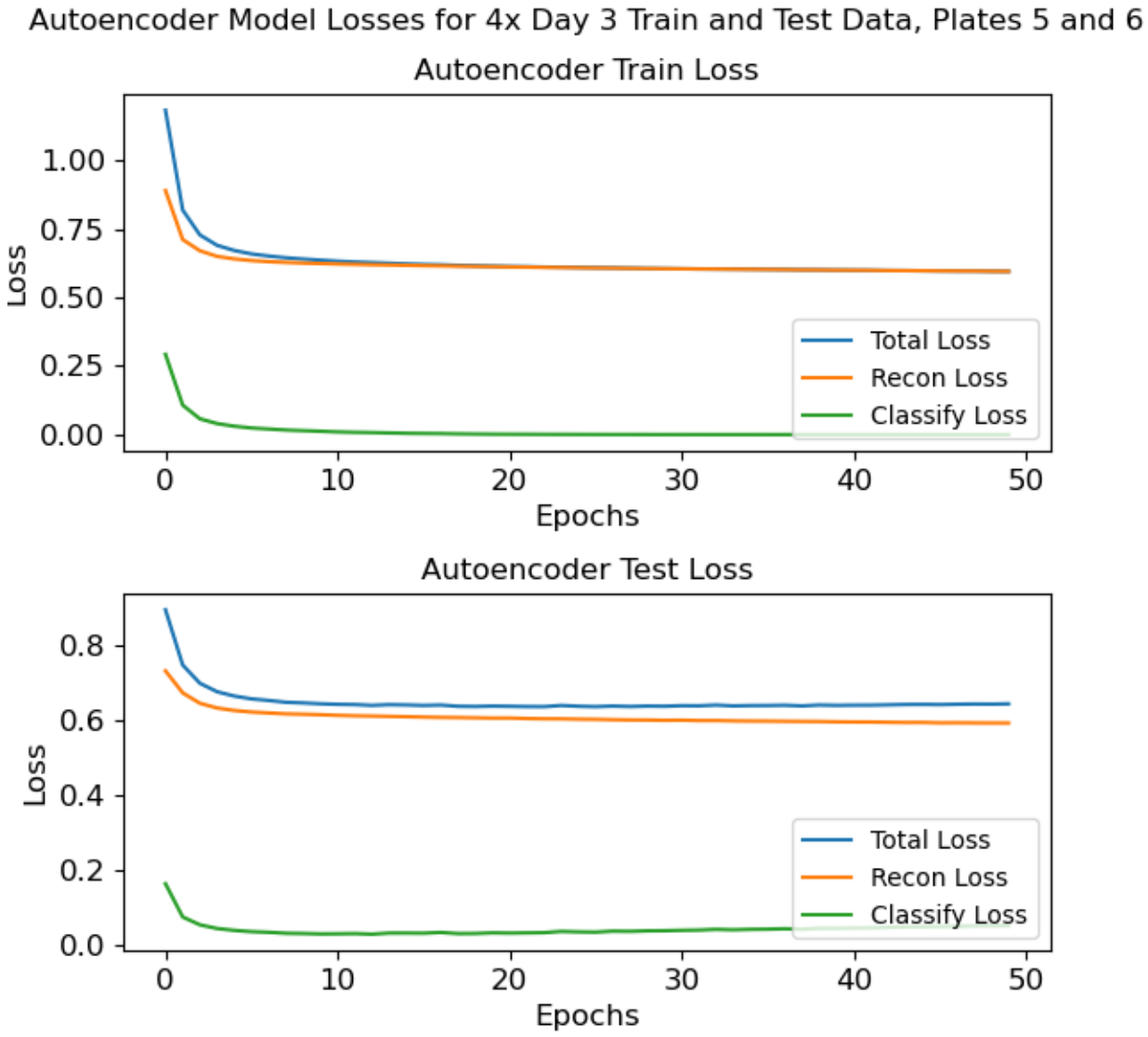
Algorithm Performances for 4x Day 3 Training and Test Data, Plates 5 and 6



ROC Curve for 4x for Day 3 Training and Test Data, Plates 5 and 6



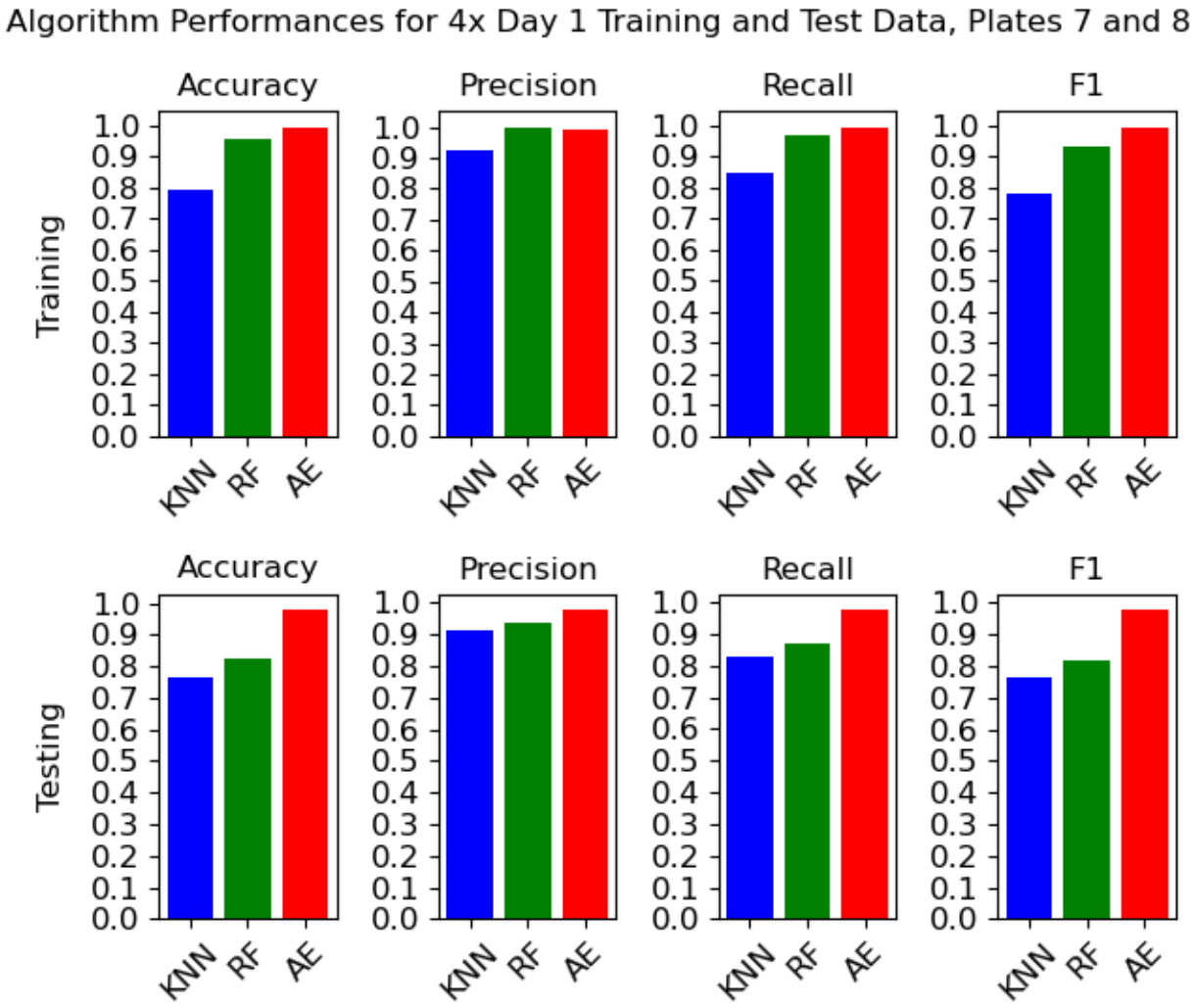
Confusion Matrices for 4x Day 3 Training and Test Data, Plates 5 and 6



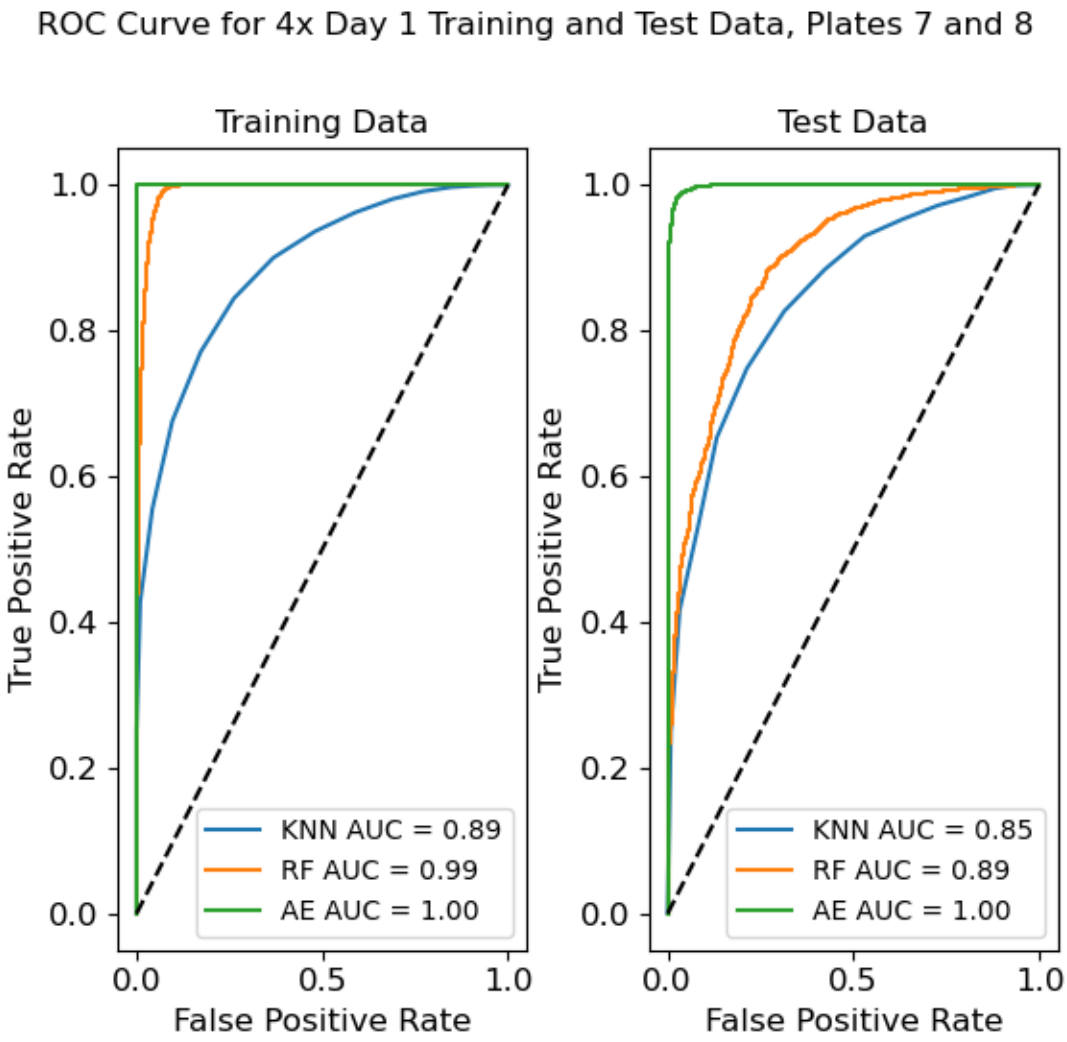
Autoencoder Model Losses for 4x Day 3 Train and Test Data, Plates 5 and 6

Plates 7 and 8 4x Data Figures

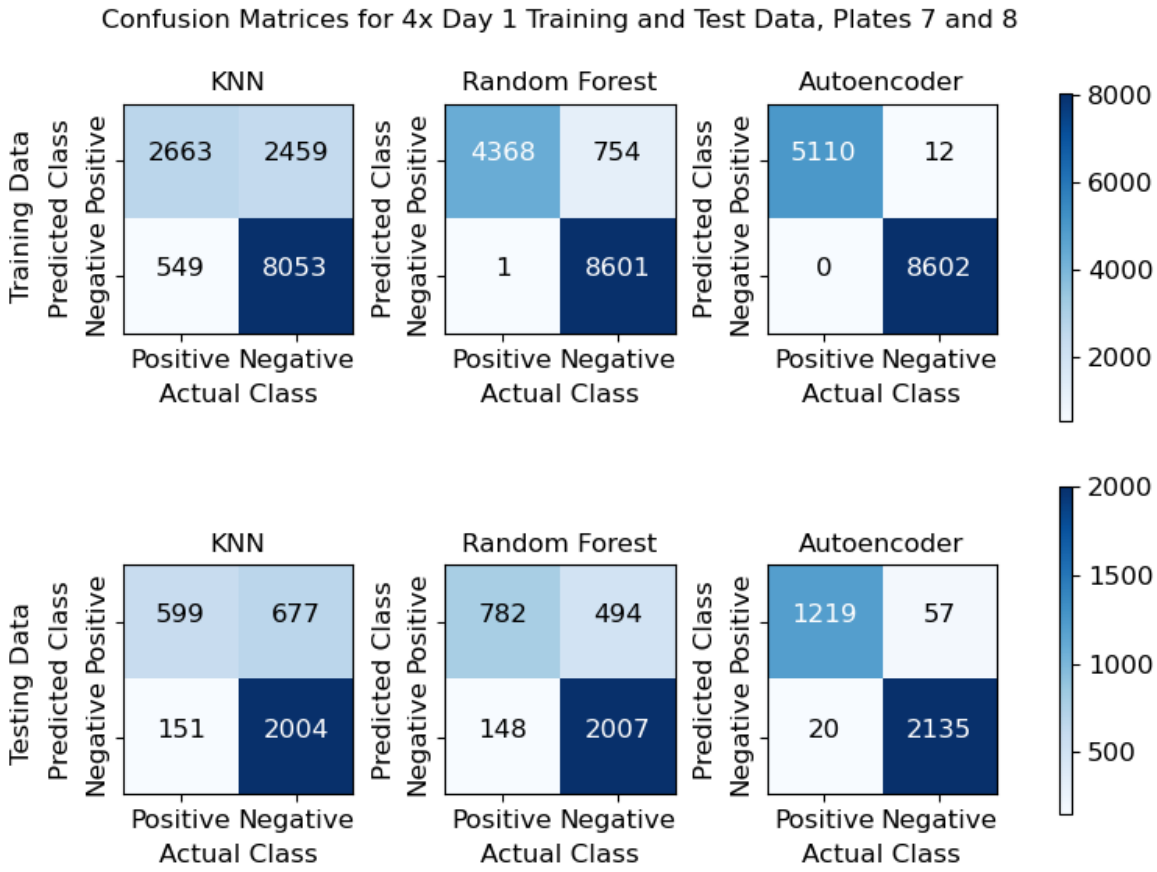
Plates 7 and 8 Day 1



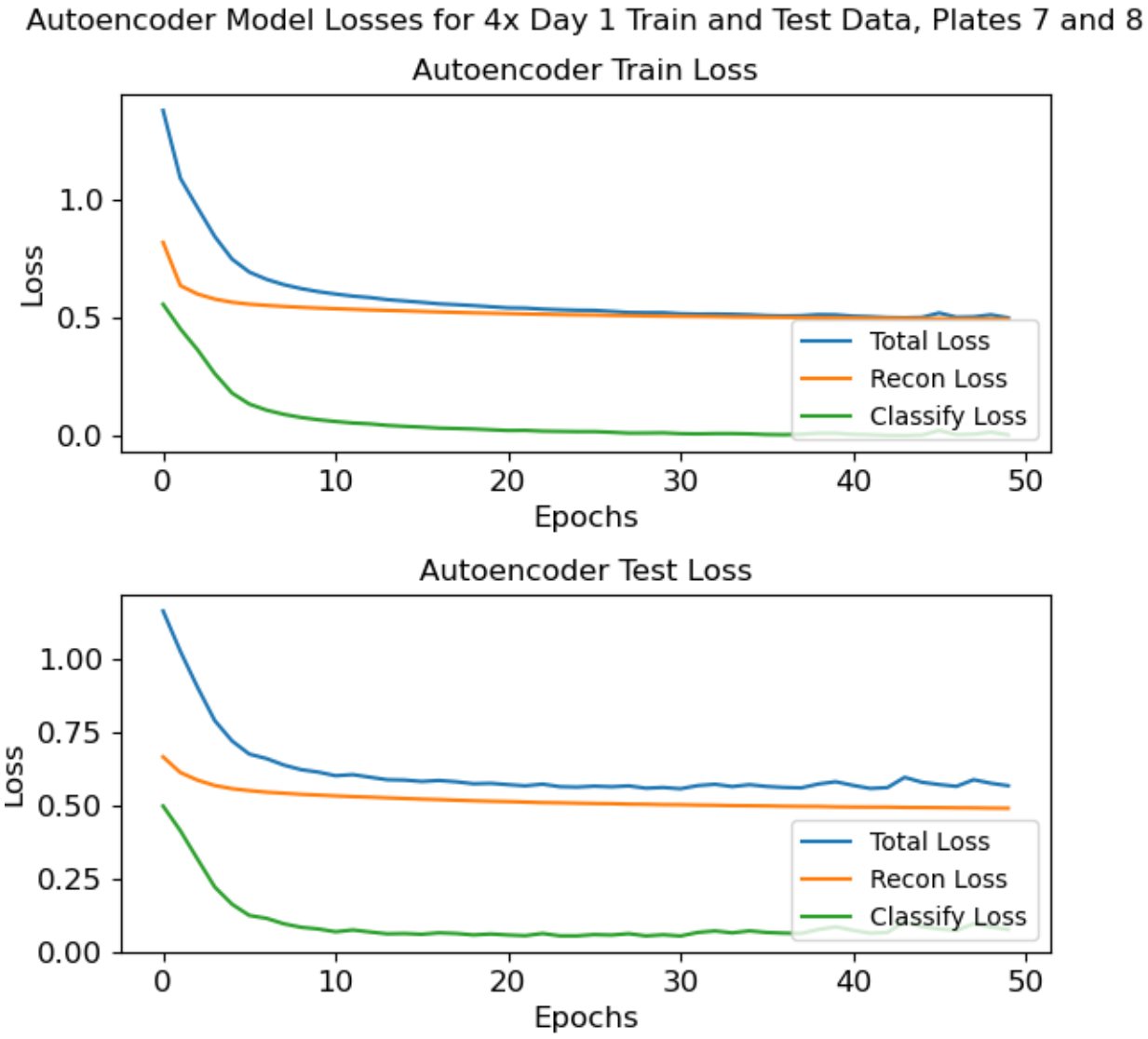
Algorithm Performances for 4x Day 1 Training and Test Data, Plates 7 and 8



ROC Curve for 4x for Day 1 Training and Test Data, Plates 7 and 8

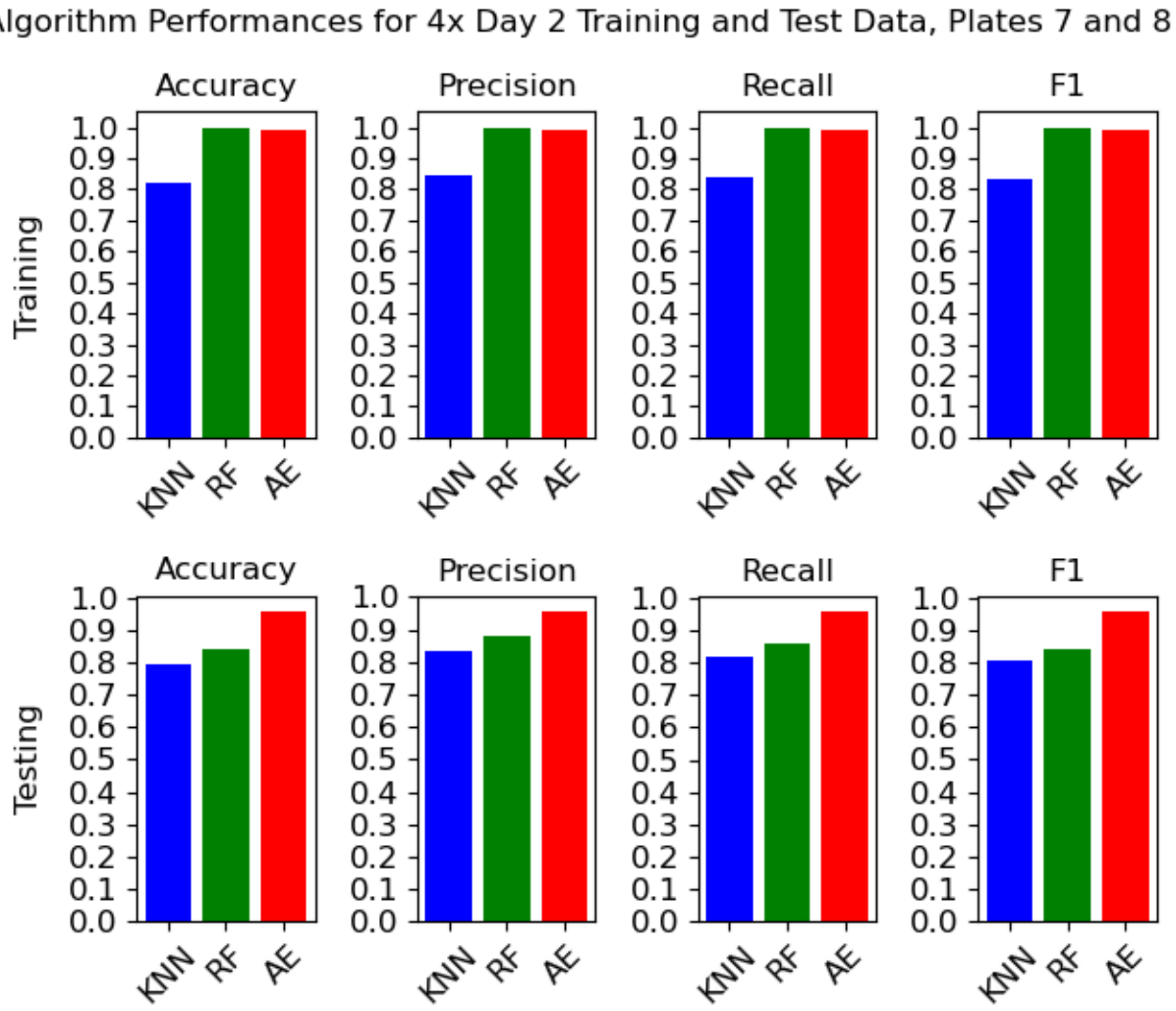


Confusion Matrices for 4x Day 1 Training and Test Data, Plates 7 and 8

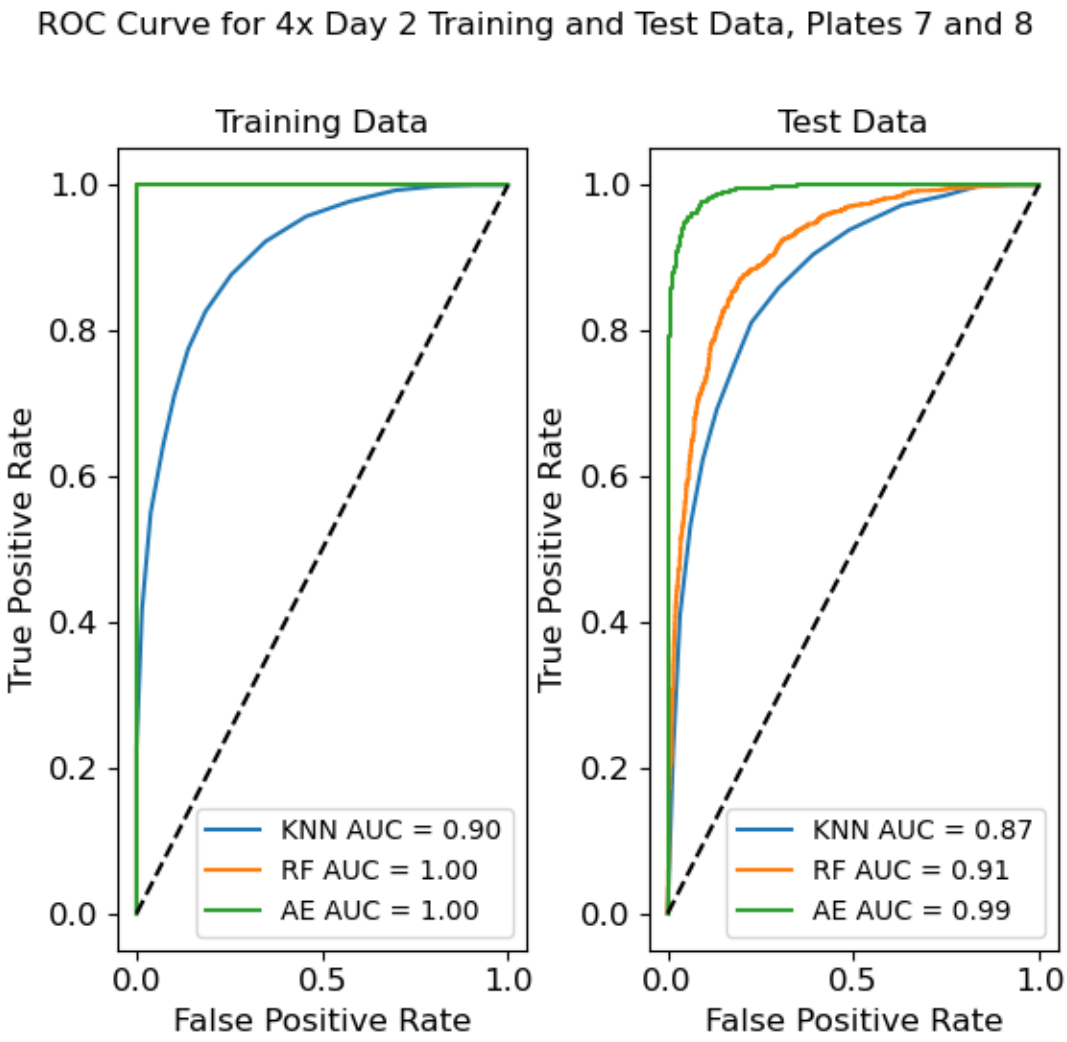


Autoencoder Model Losses for 4x Day 1 Train and Test Data, Plates 7 and 8

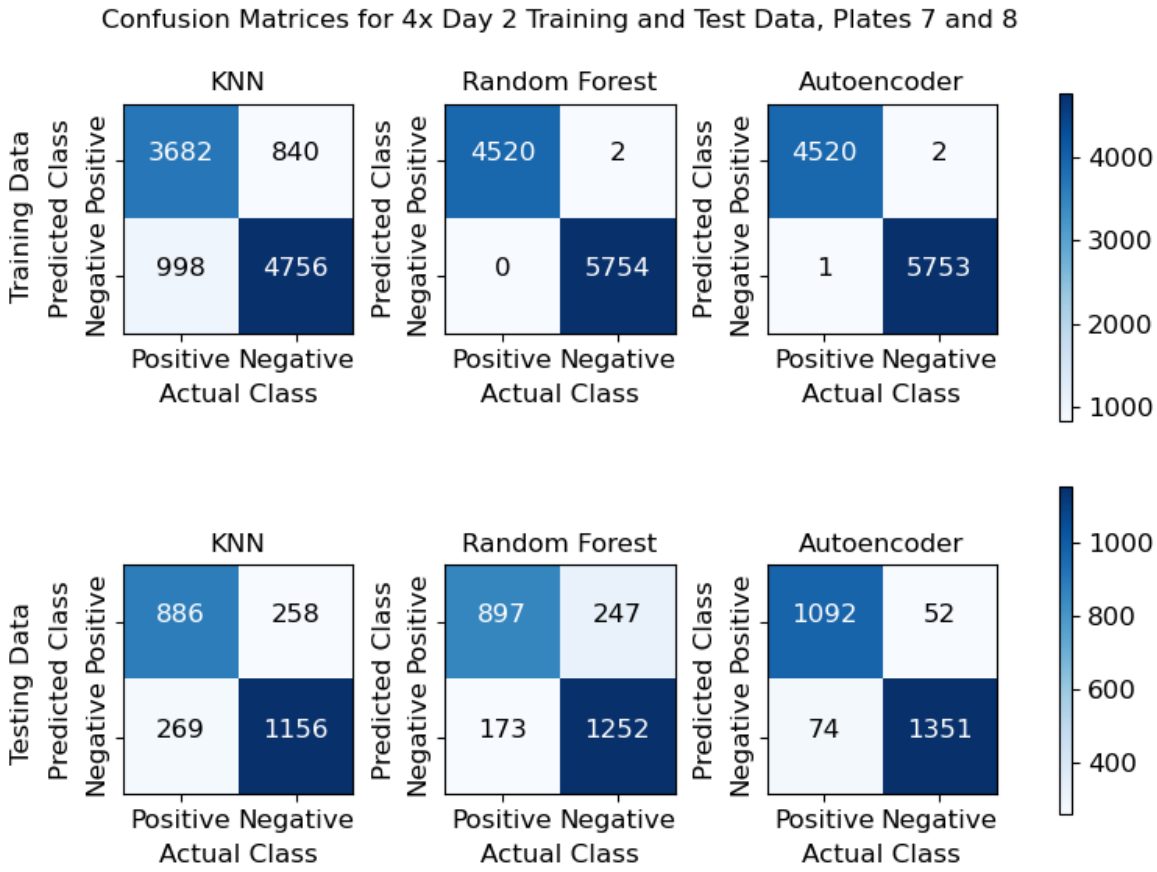
Plates 7 and 8 Day 2



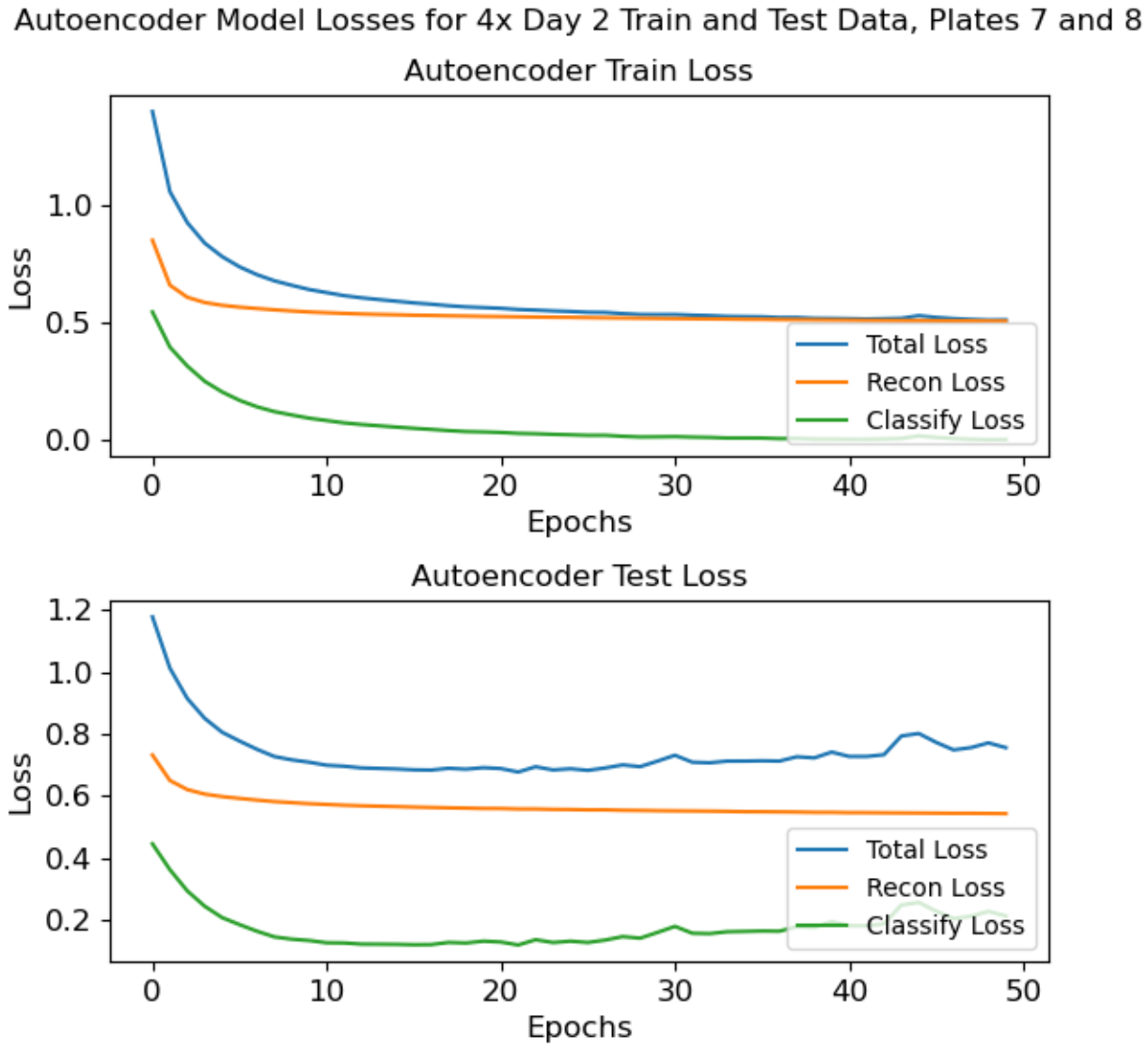
Algorithm Performances for 4x Day 2 Training and Test Data, Plates 7 and 8



ROC Curve for 4x for Day 2 Training and Test Data, Plates 7 and 8

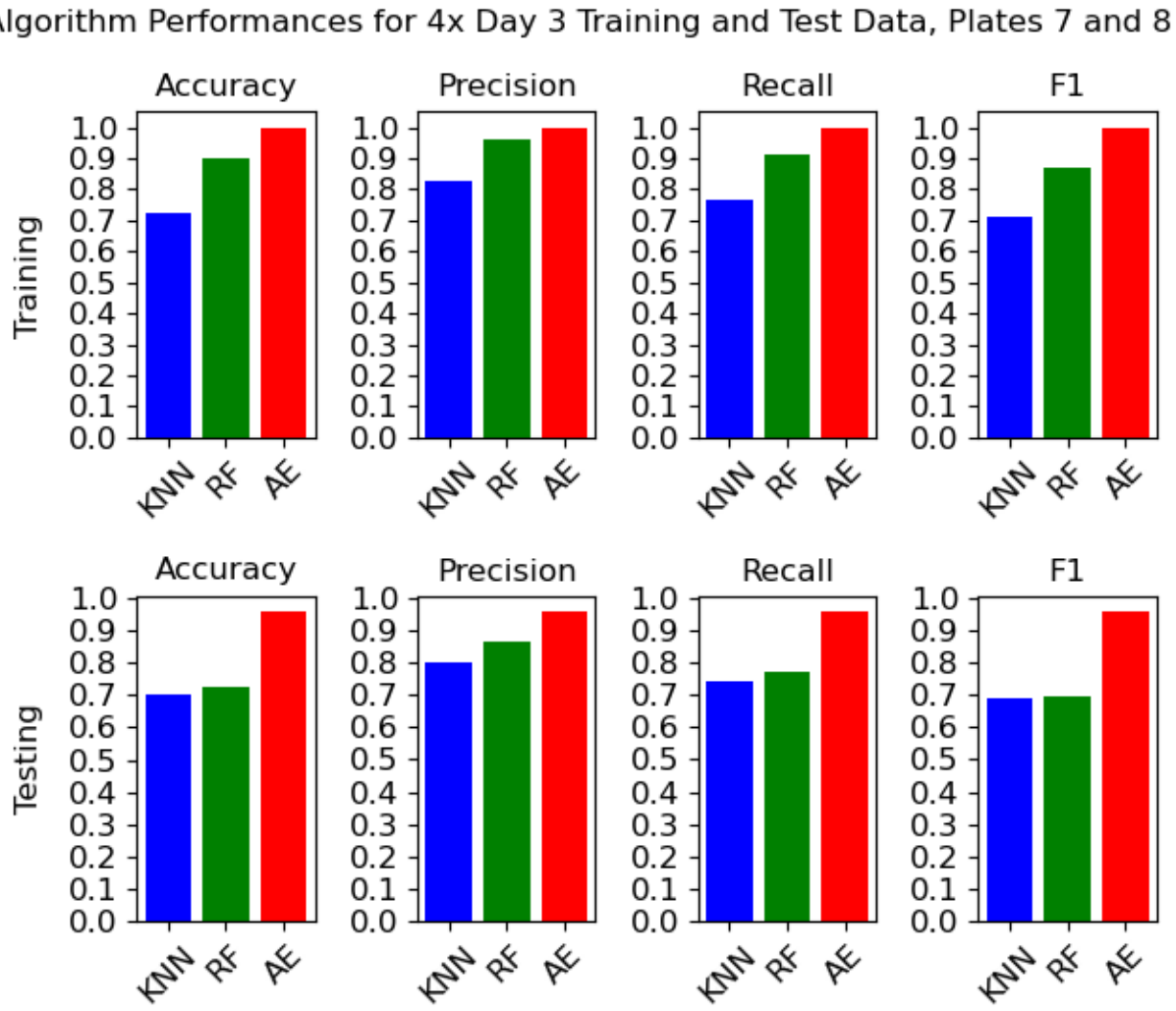


Confusion Matrices for 4x Day 2 Training and Test Data, Plates 7 and 8

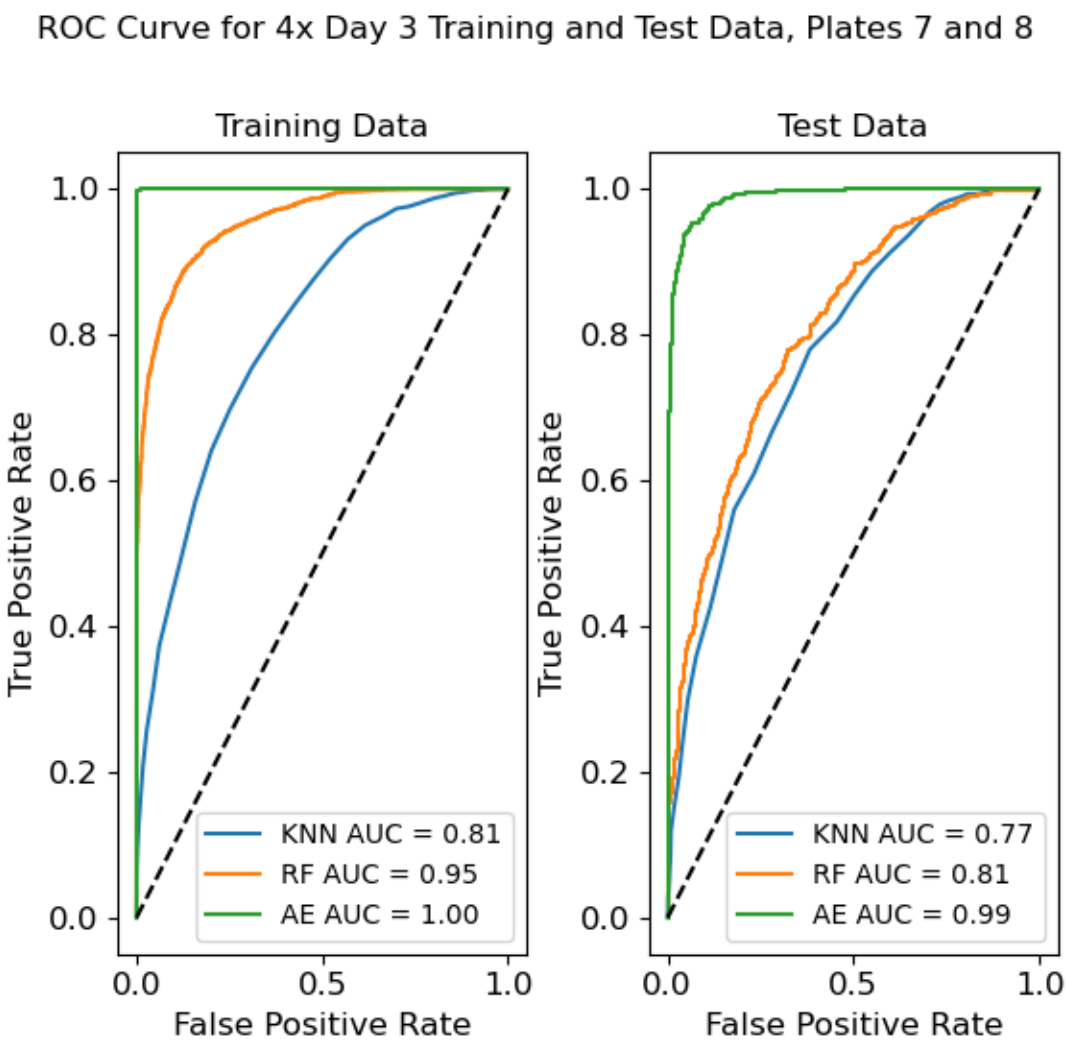


Autoencoder Model Losses for 4x Day 2 Train and Test Data, Plates 7 and 8

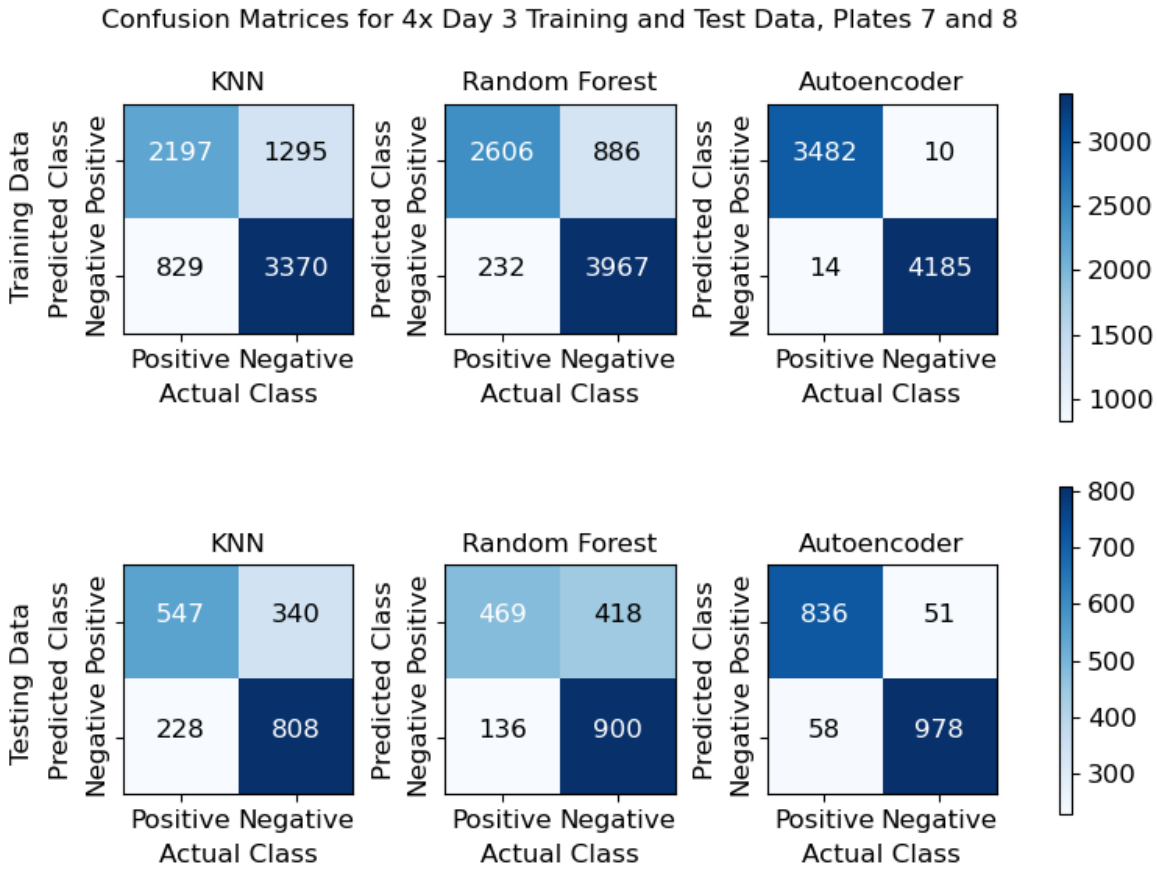
Plates 7 and 8 Day 3



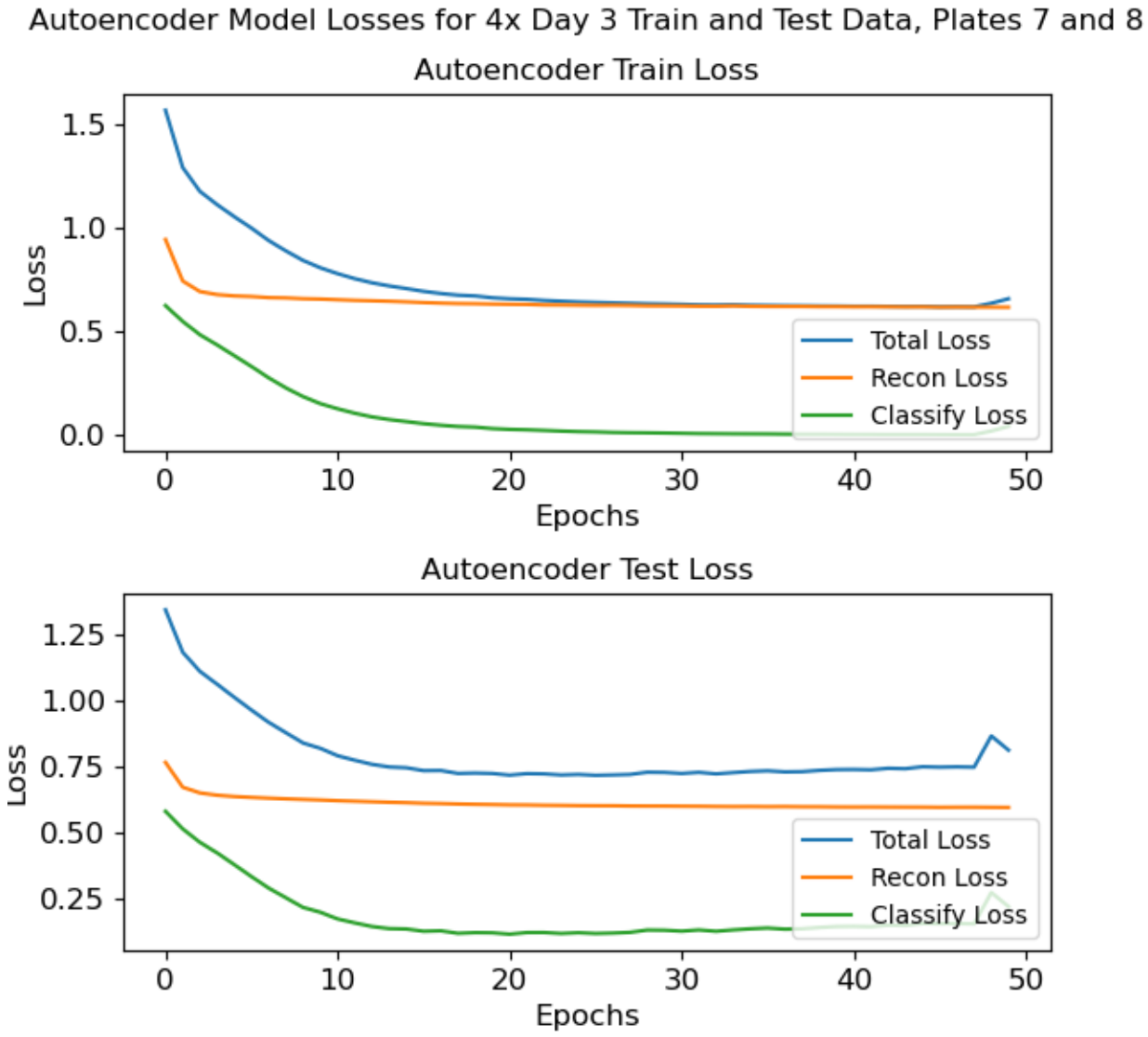
Algorithm Performances for 4x Day 3 Training and Test Data, Plates 7 and 8



ROC Curve for 4x for Day 3 Training and Test Data, Plates 7 and 8



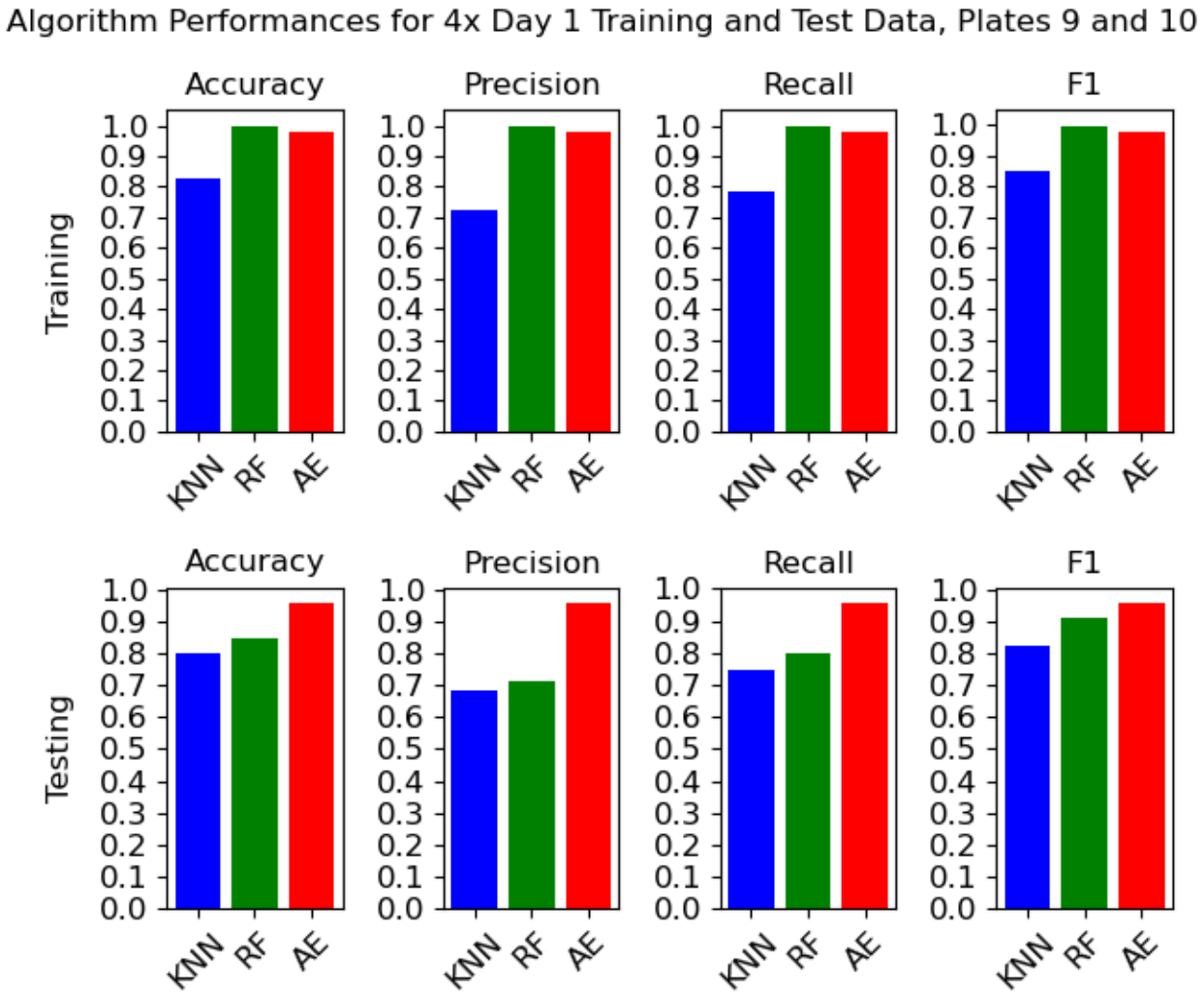
Confusion Matrices for 4x Day 3 Training and Test Data, Plates 7 and 8



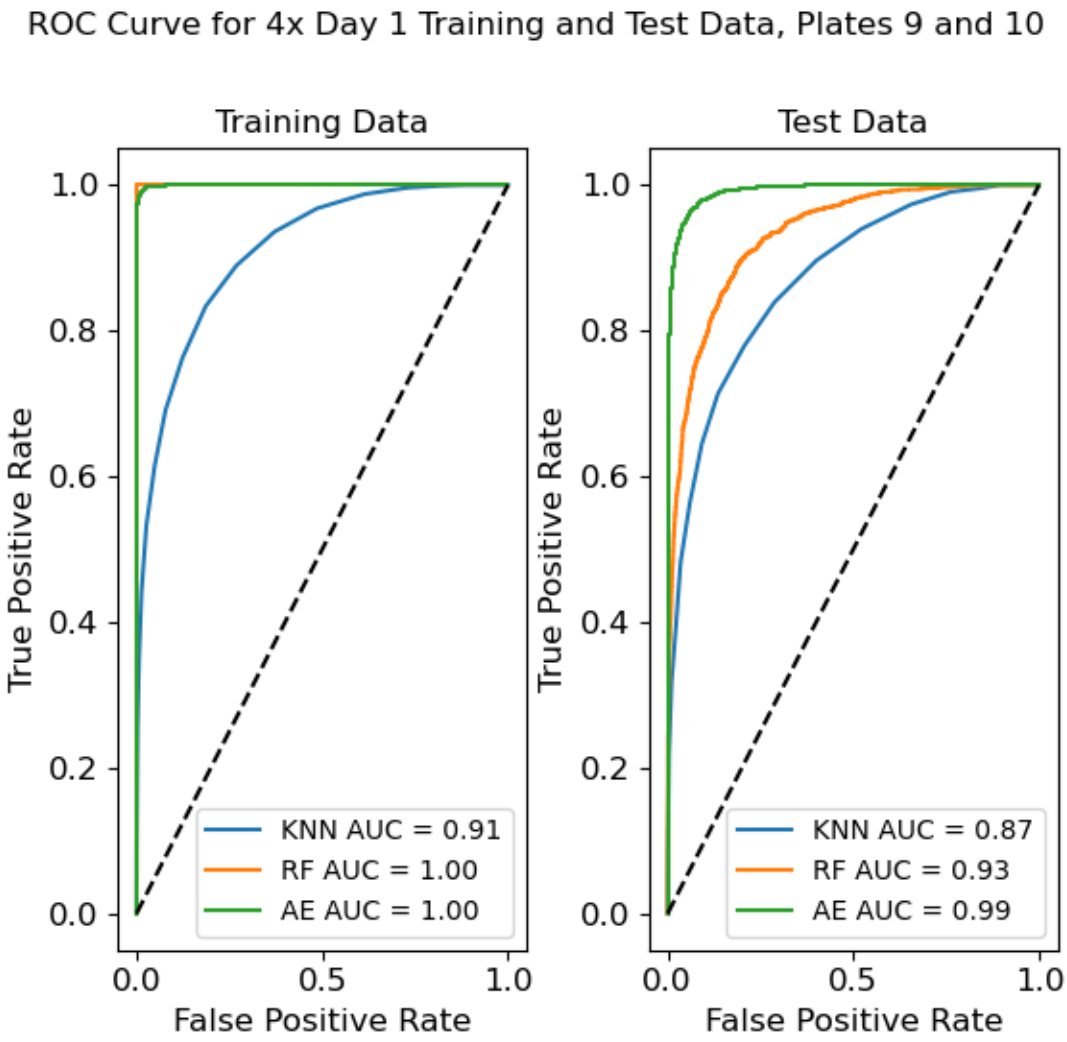
Autoencoder Model Losses for 4x Day 3 Train and Test Data, Plates 7 and 8

Plates 9 and 10 4x Data Figures

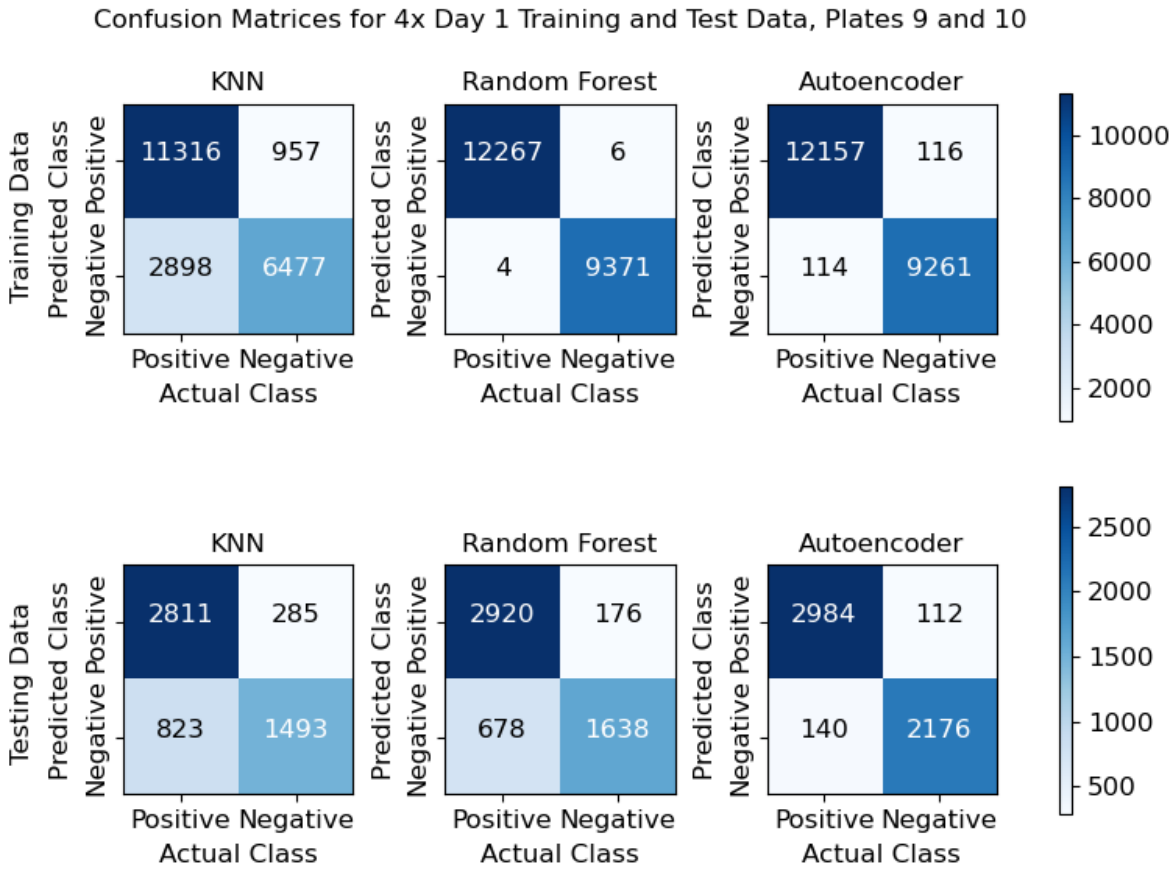
Plates 9 and 10 Day 1



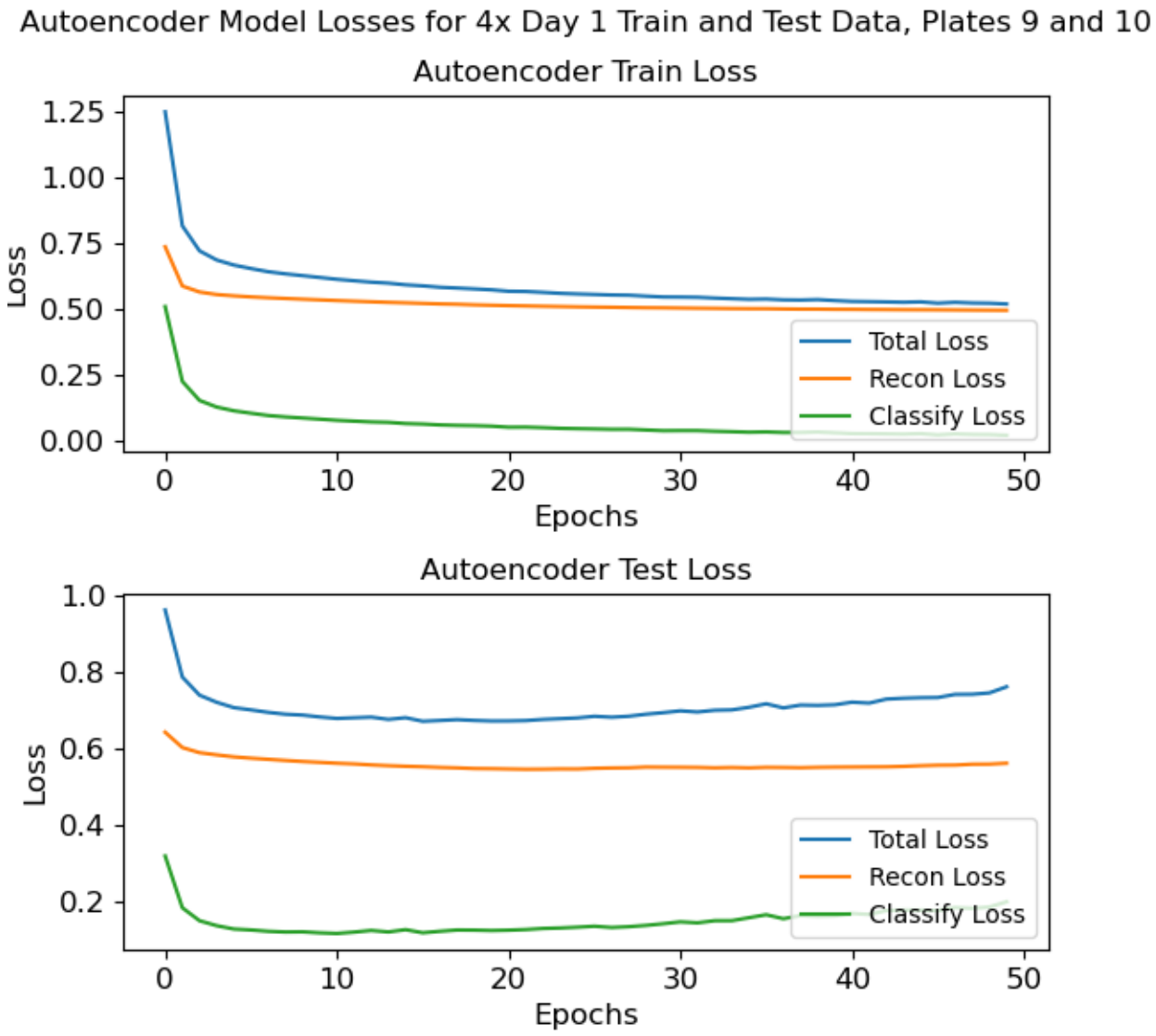
Algorithm Performances for 4x Day 1 Training and Test Data, Plates 9 and 10



ROC Curve for 4x for Day 1 Training and Test Data, Plates 9 and 10

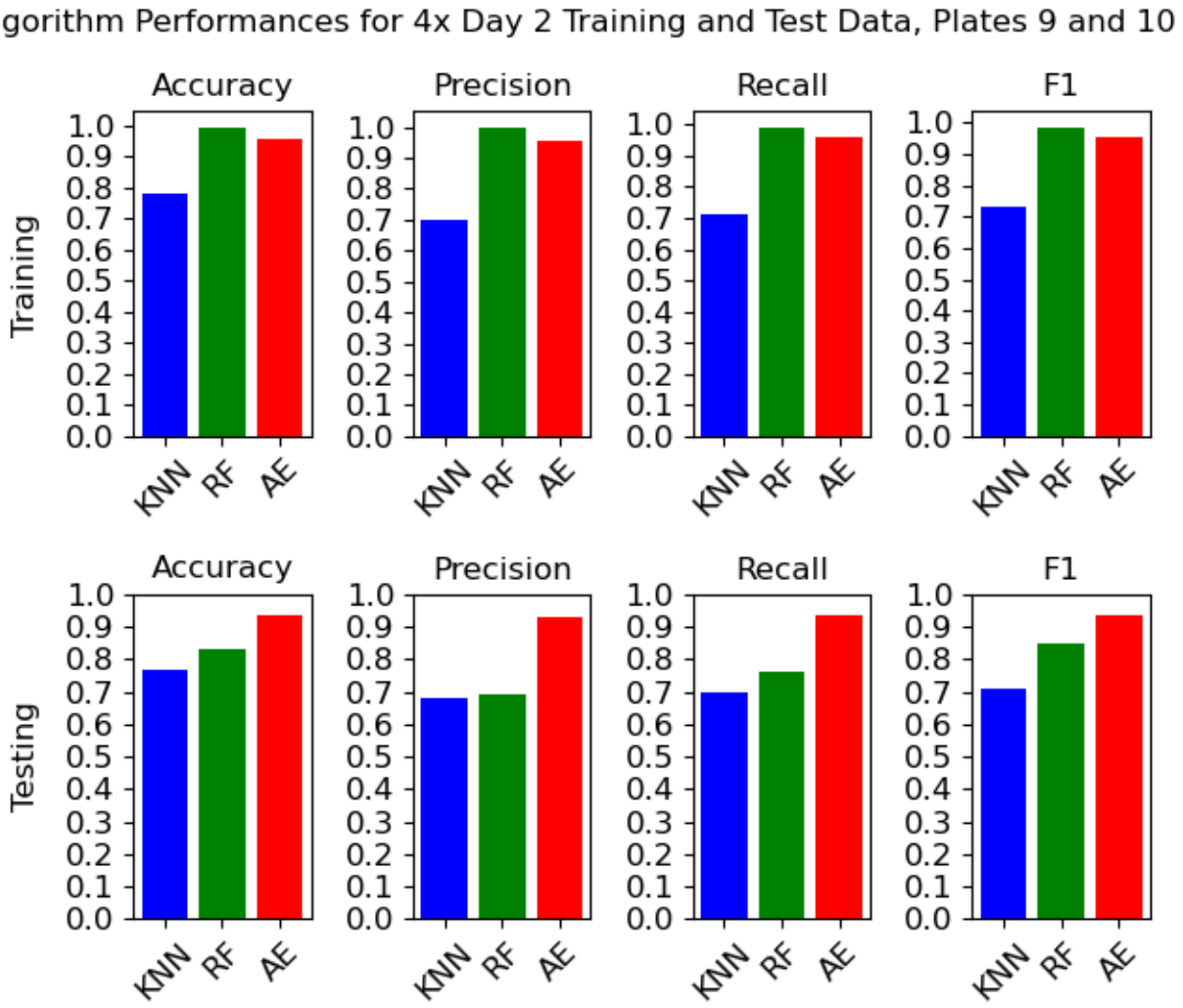


Confusion Matrices for 4x Day 1 Training and Test Data, Plates 9 and 10

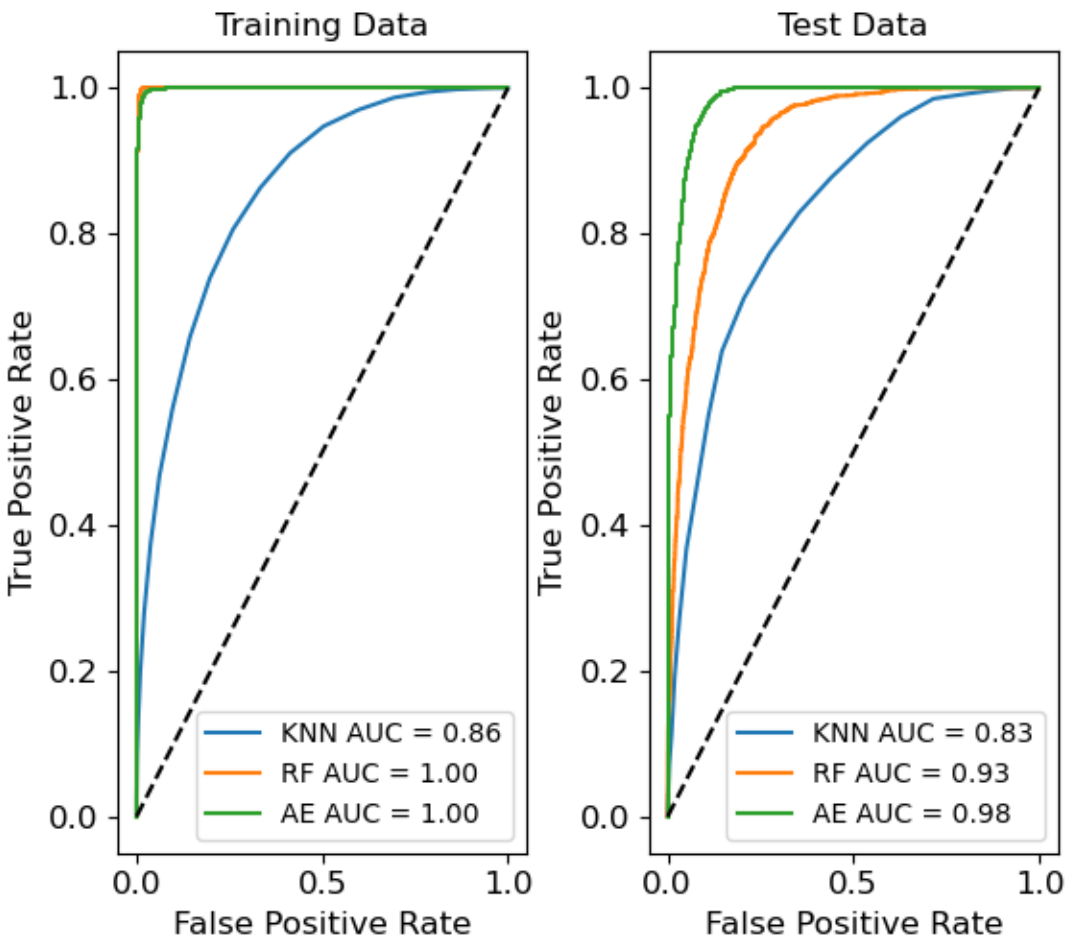


Autoencoder Model Losses for 4x Day 1 Train and Test Data, Plates 9 and 10

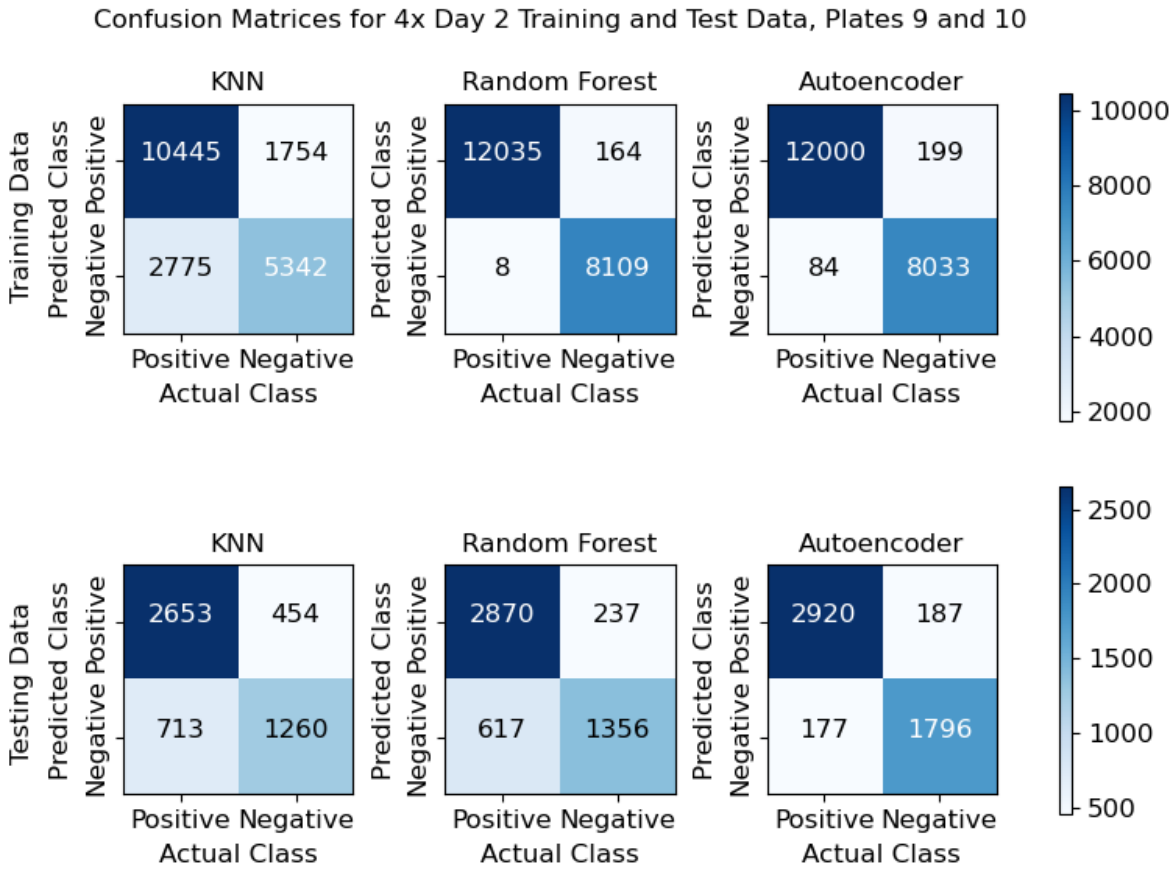
Plates 9 and 10 Day 2



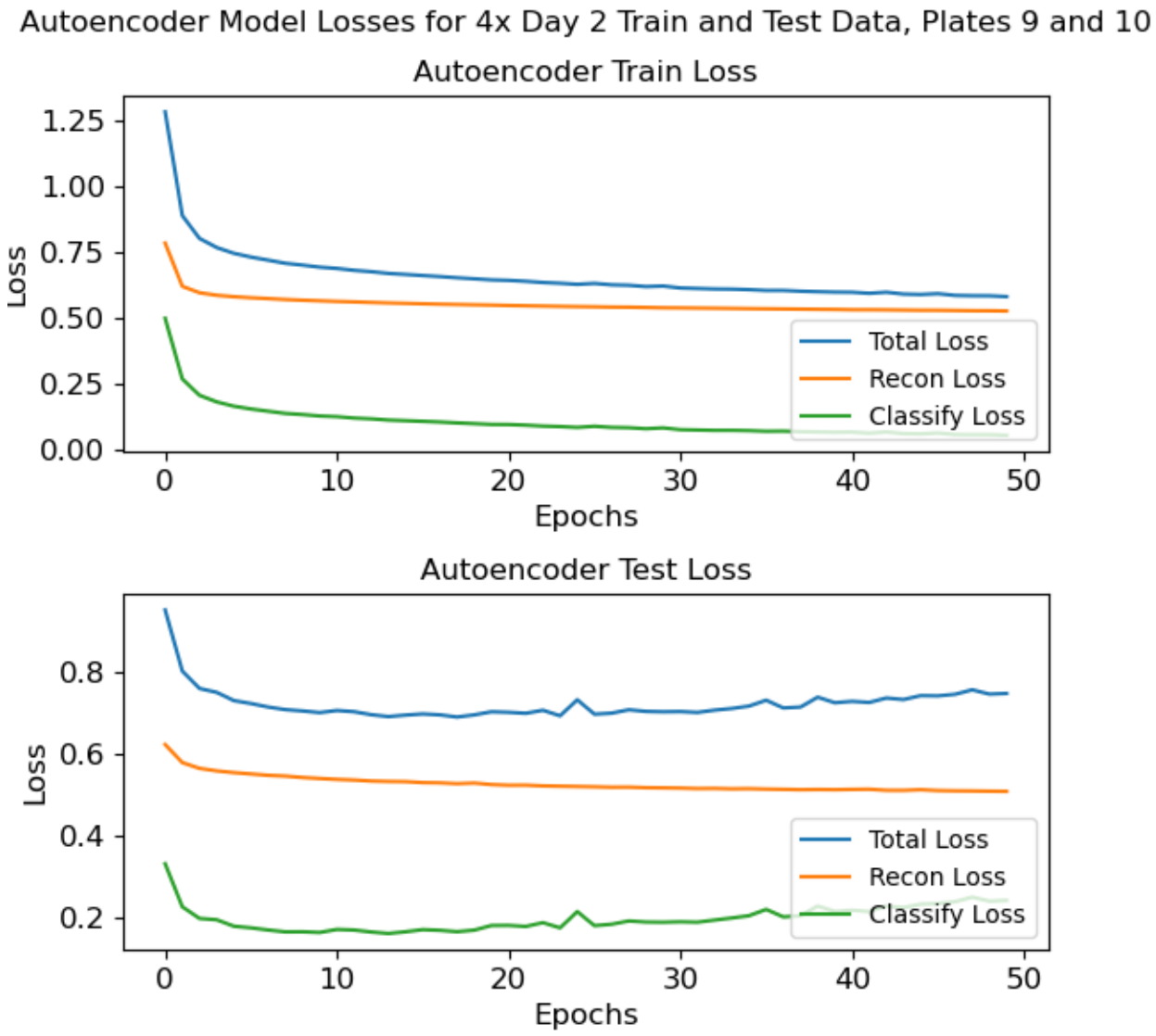
Algorithm Performances for 4x Day 2 Training and Test Data, Plates 9 and 10



ROC Curve for 4x for Day 2 Training and Test Data, Plates 9 and 10



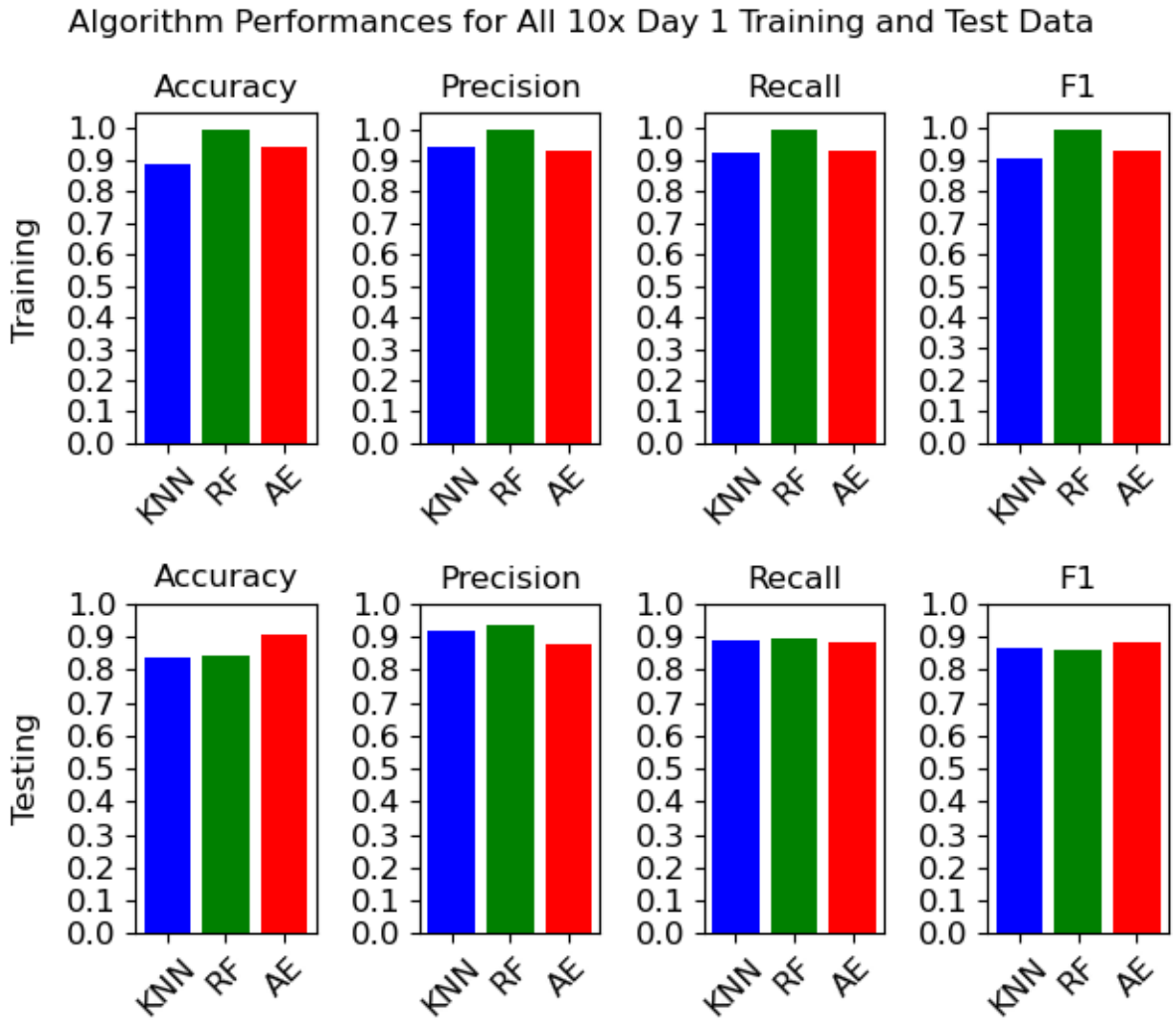
Confusion Matrices for 4x Day 2 Training and Test Data, Plates 9 and 10



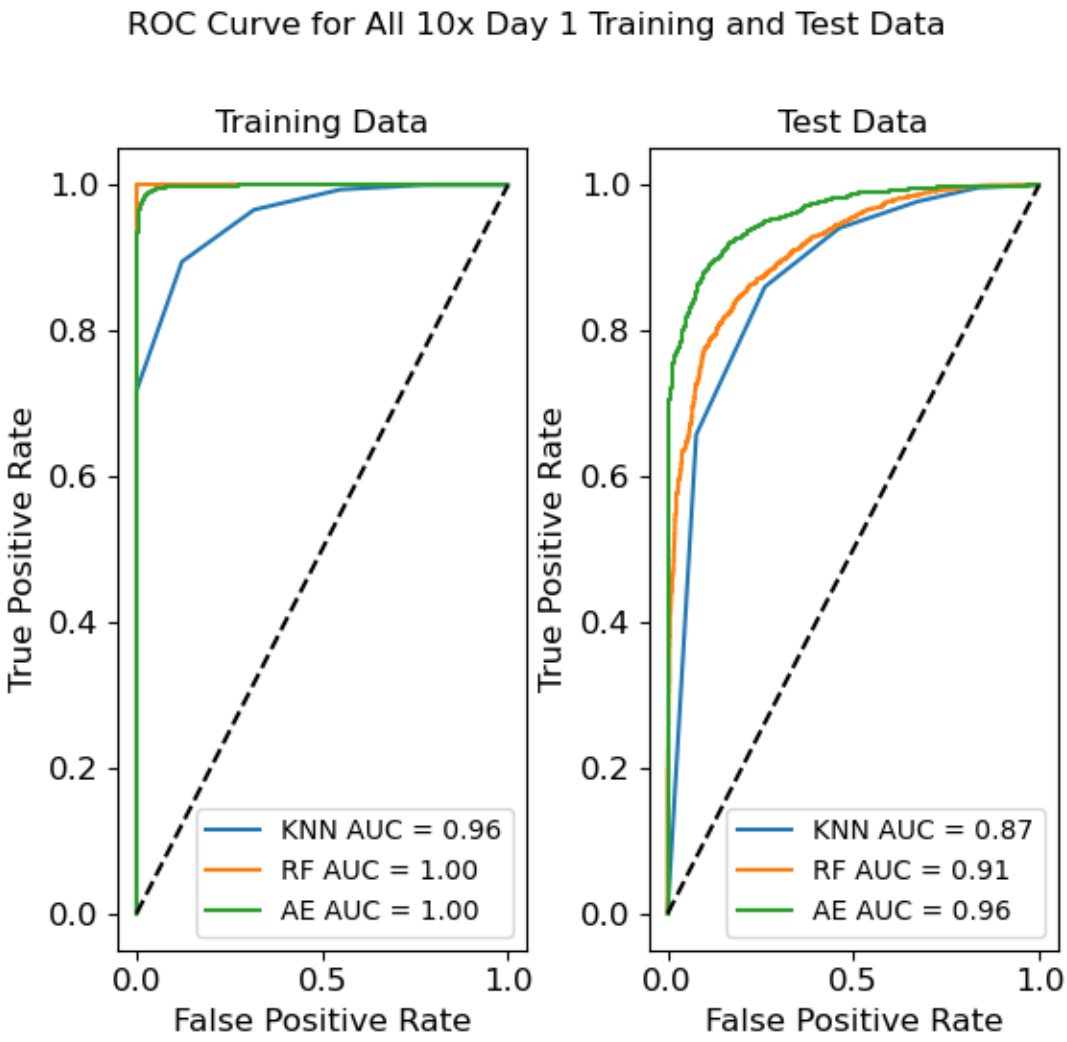
Autoencoder Model Losses for 4x Day 2 Train and Test Data, Plates 9 and 10

Aggregate 10x Data Figures

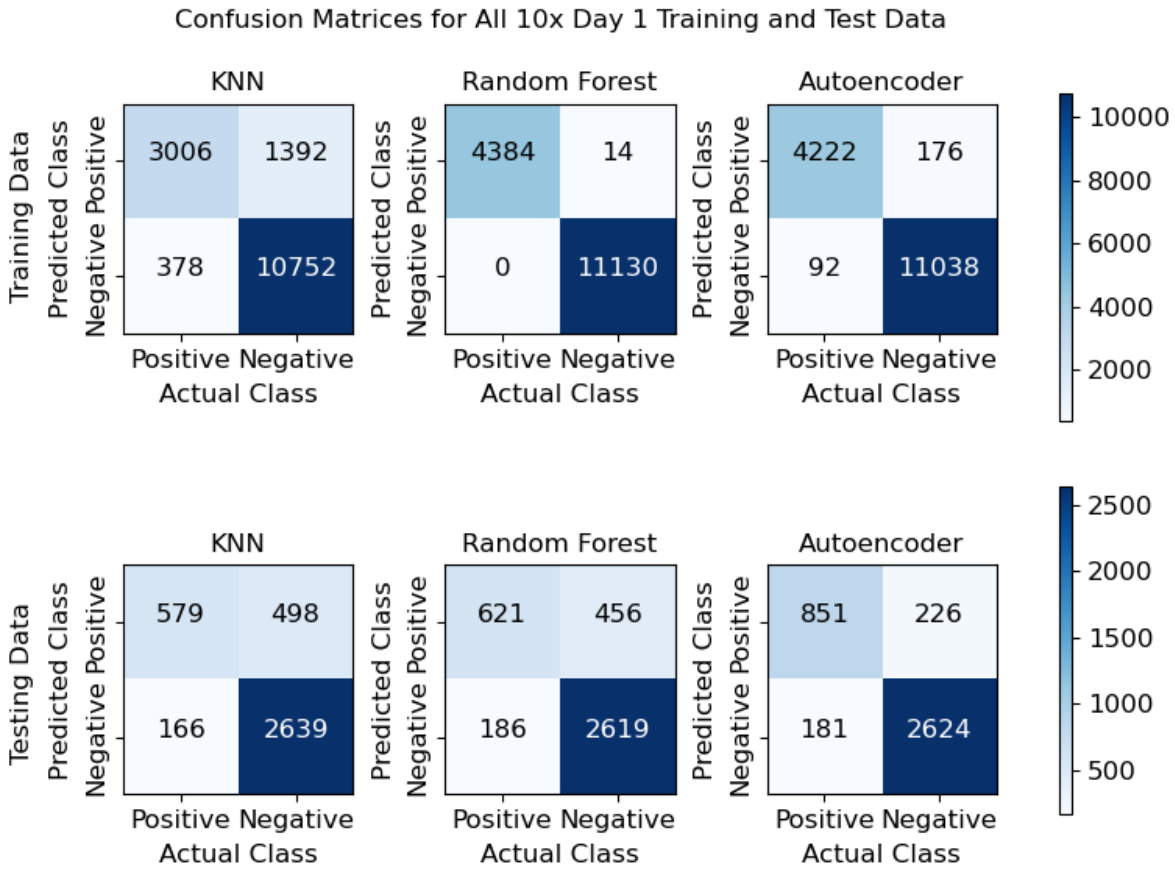
Aggregate Data Evaluation 10x Day 1



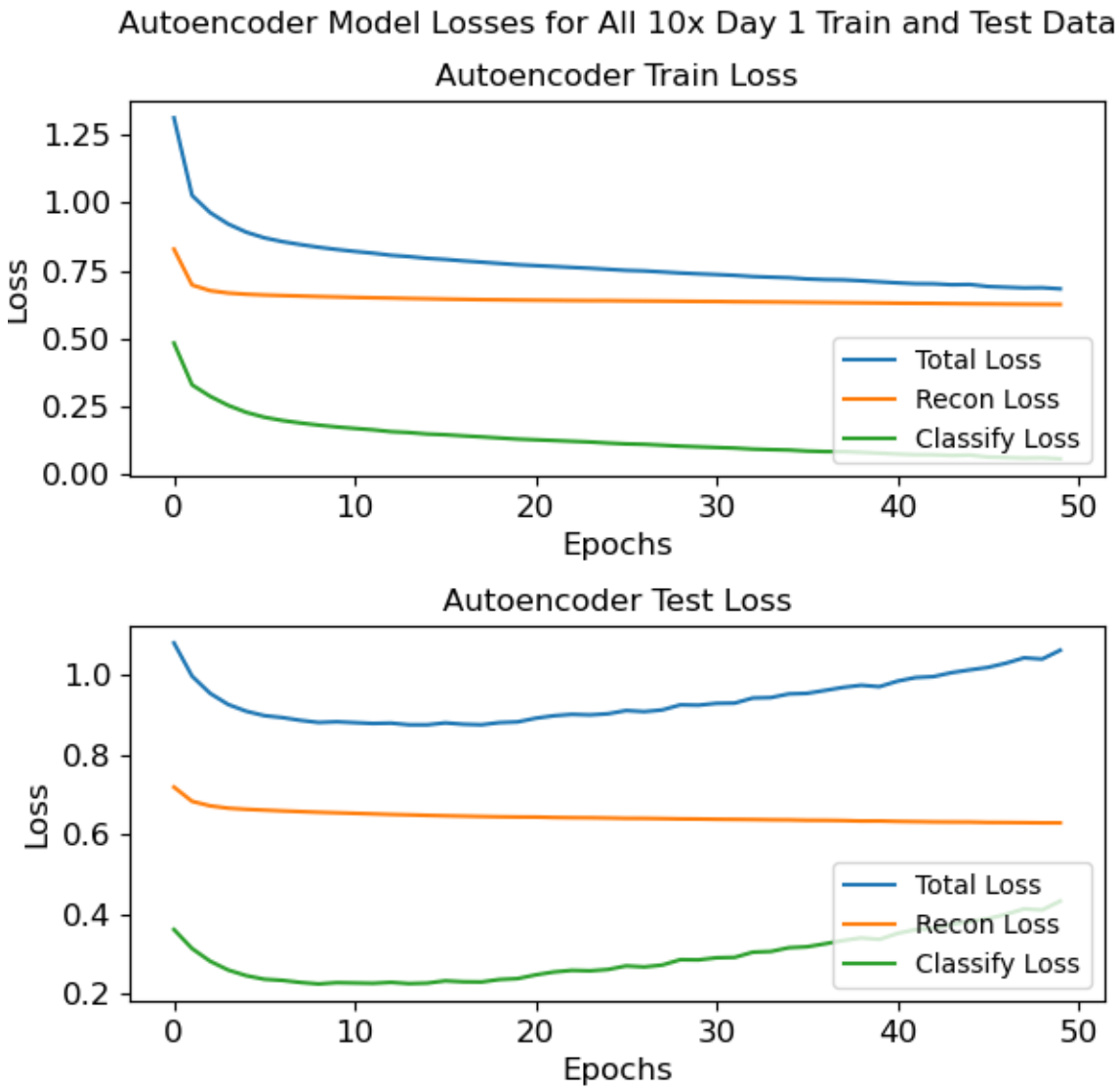
Algorithm Performances for All 10x Day 1 Training and Test Data



ROC Curve for All 10x Day 1 Training and Test Data

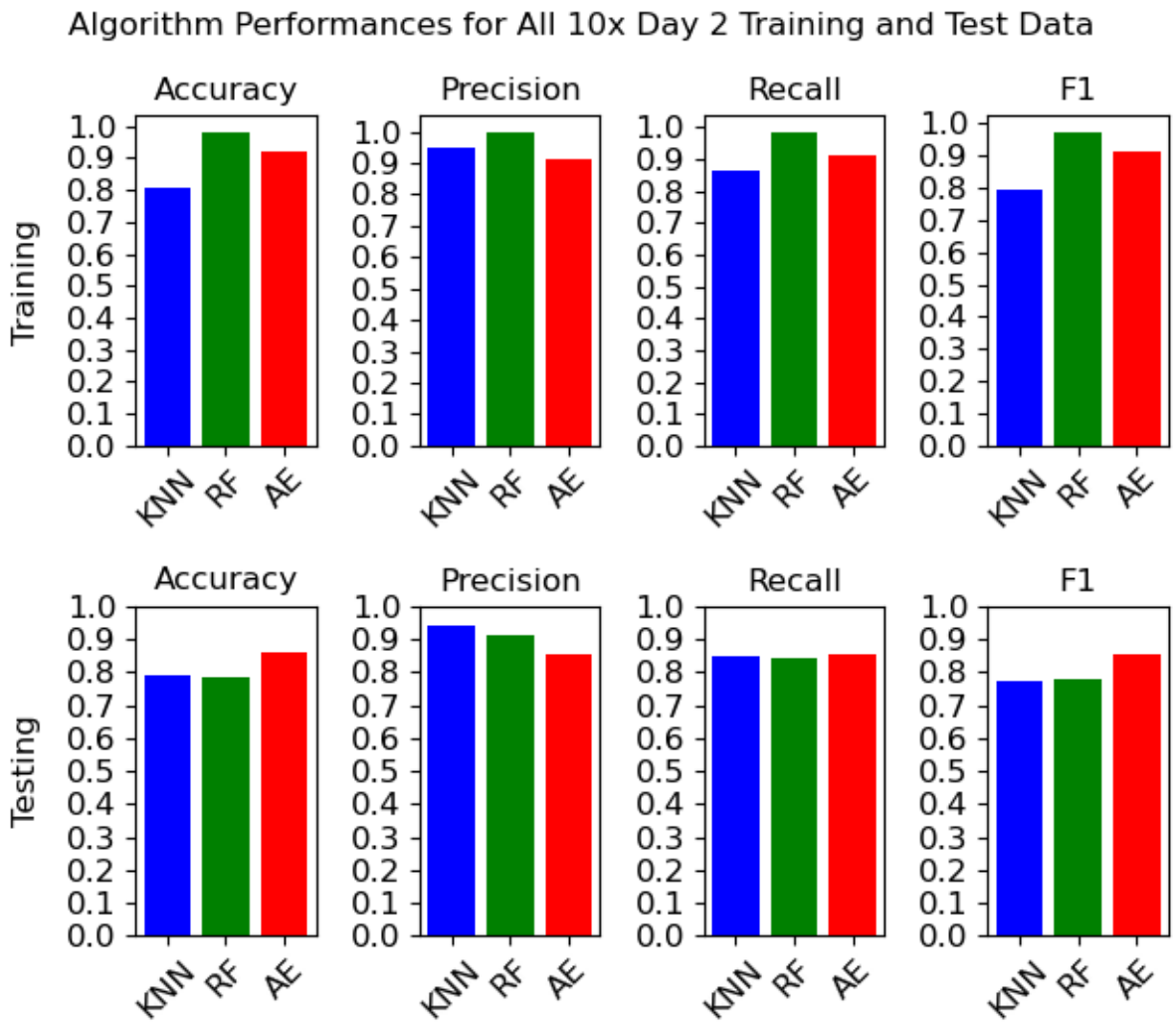


Confusion Matrices for All 10x Day 1 Training and Test Data

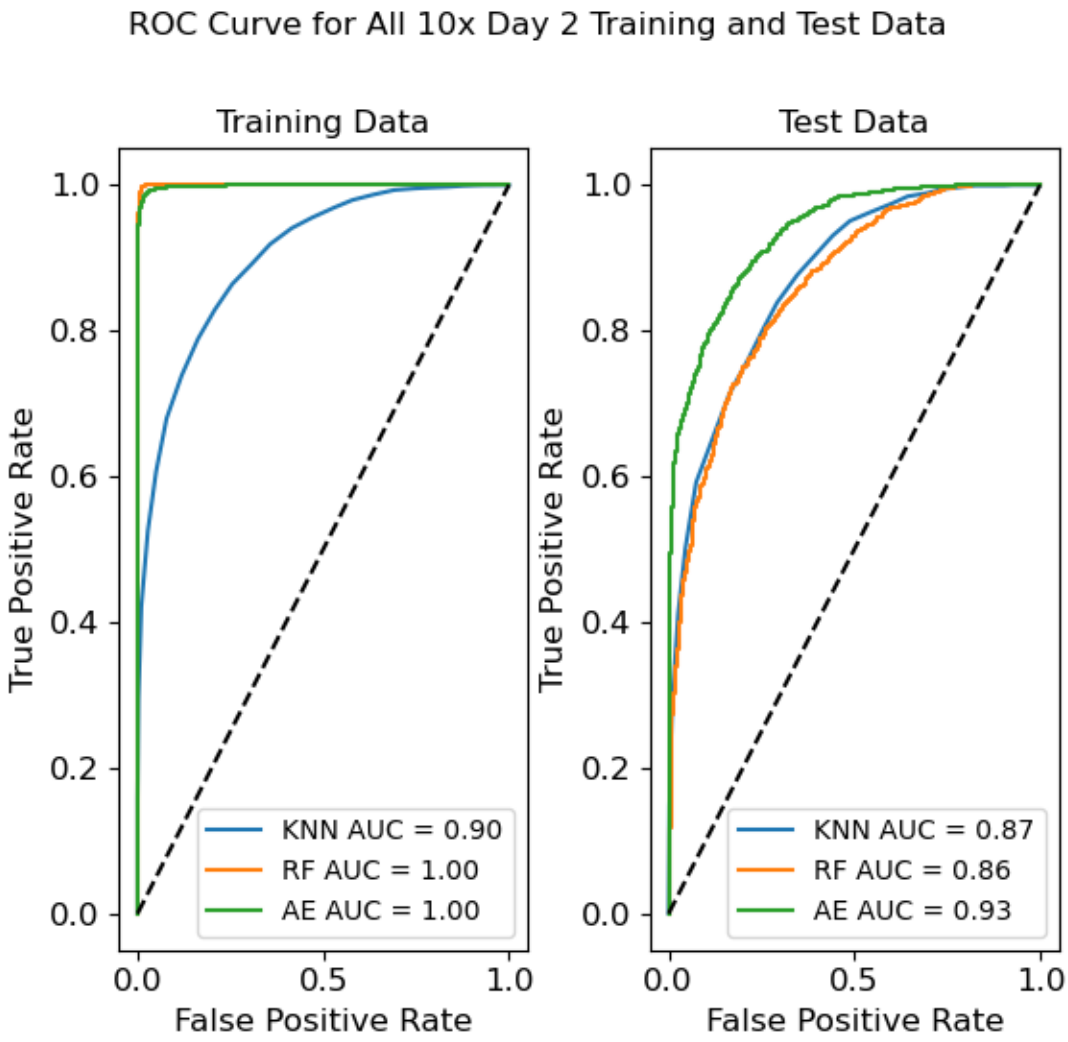


Autoencoder Model Losses for All 10x Day 1 Train and Test Data

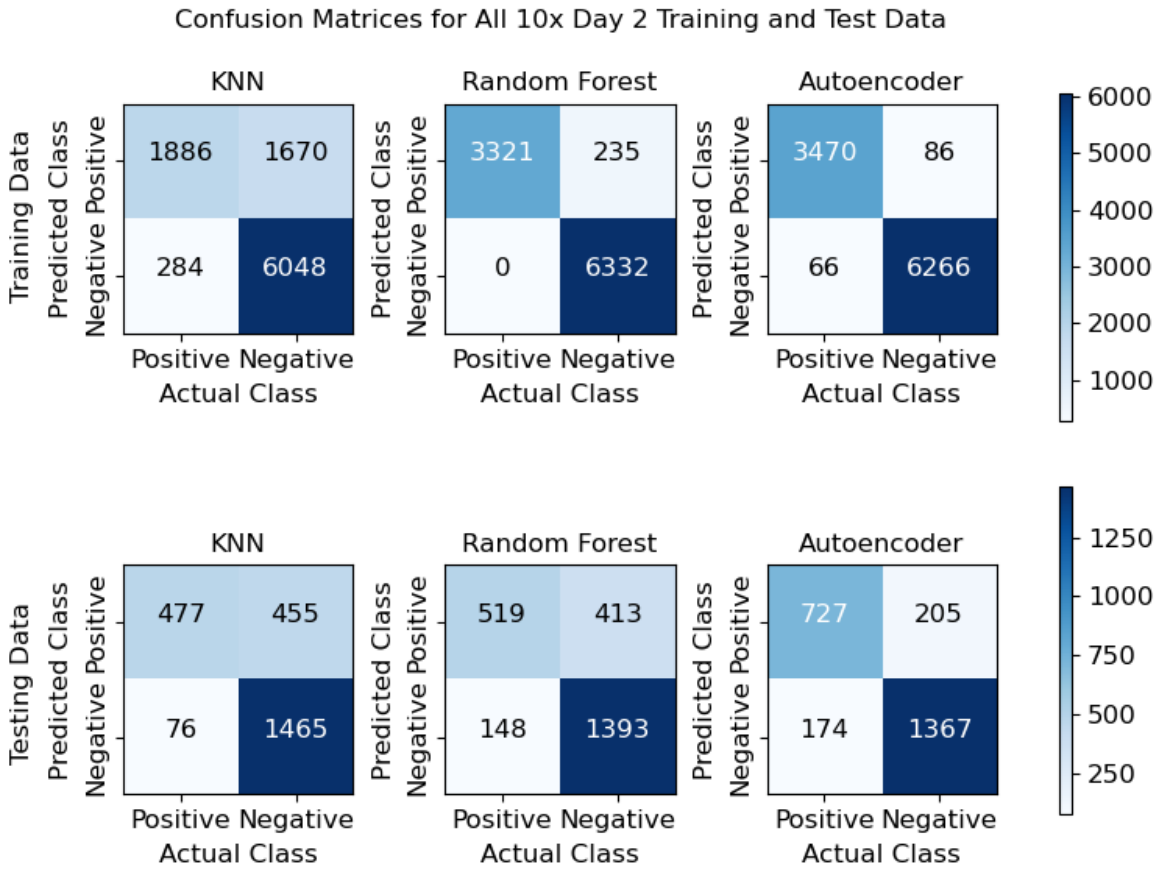
Aggregate Data Evaluation 10x Day 2



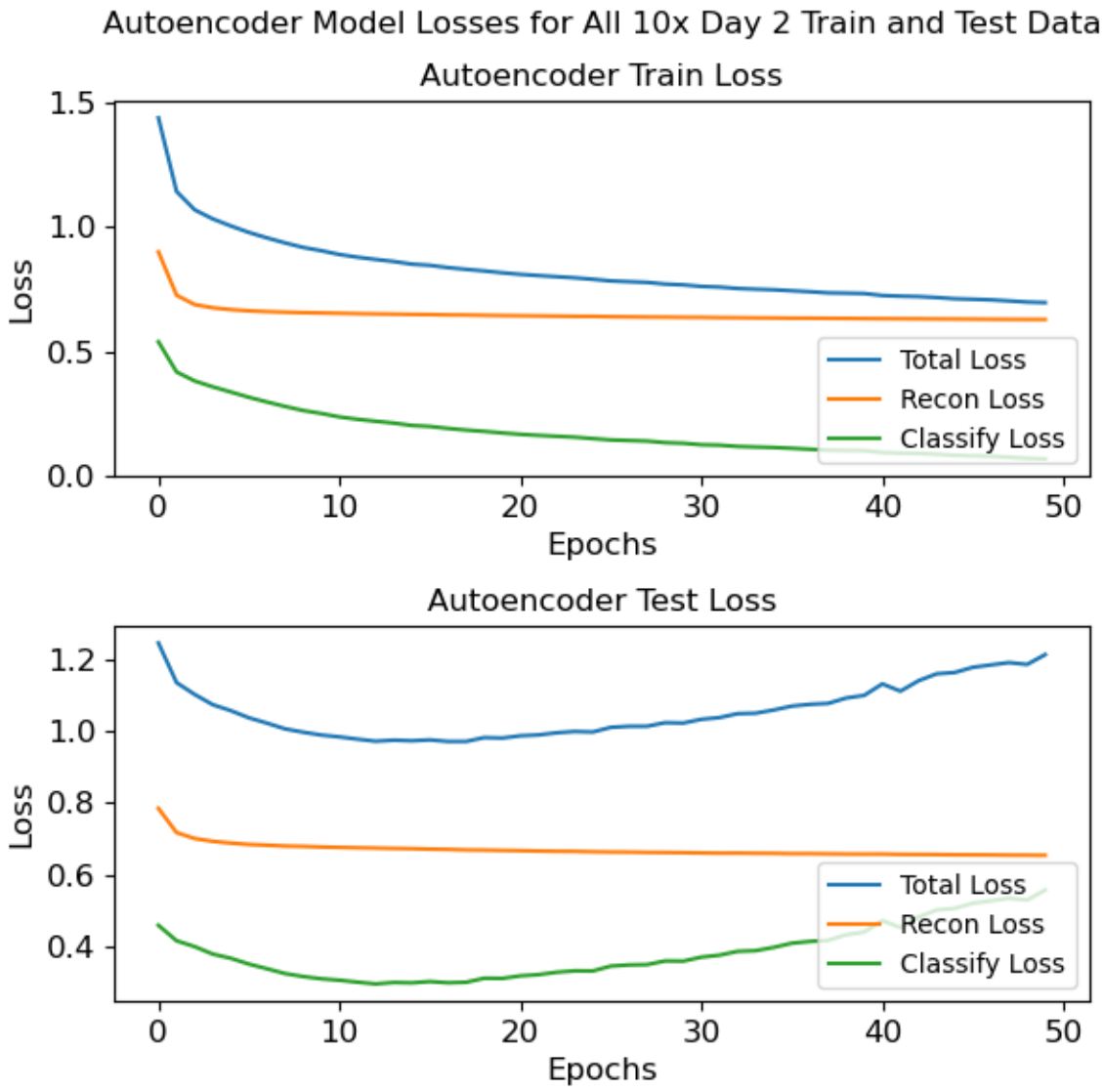
Algorithm Performances for All 10x Day 2 Training and Test Data



ROC Curve for All 10x Day 2 Training and Test Data

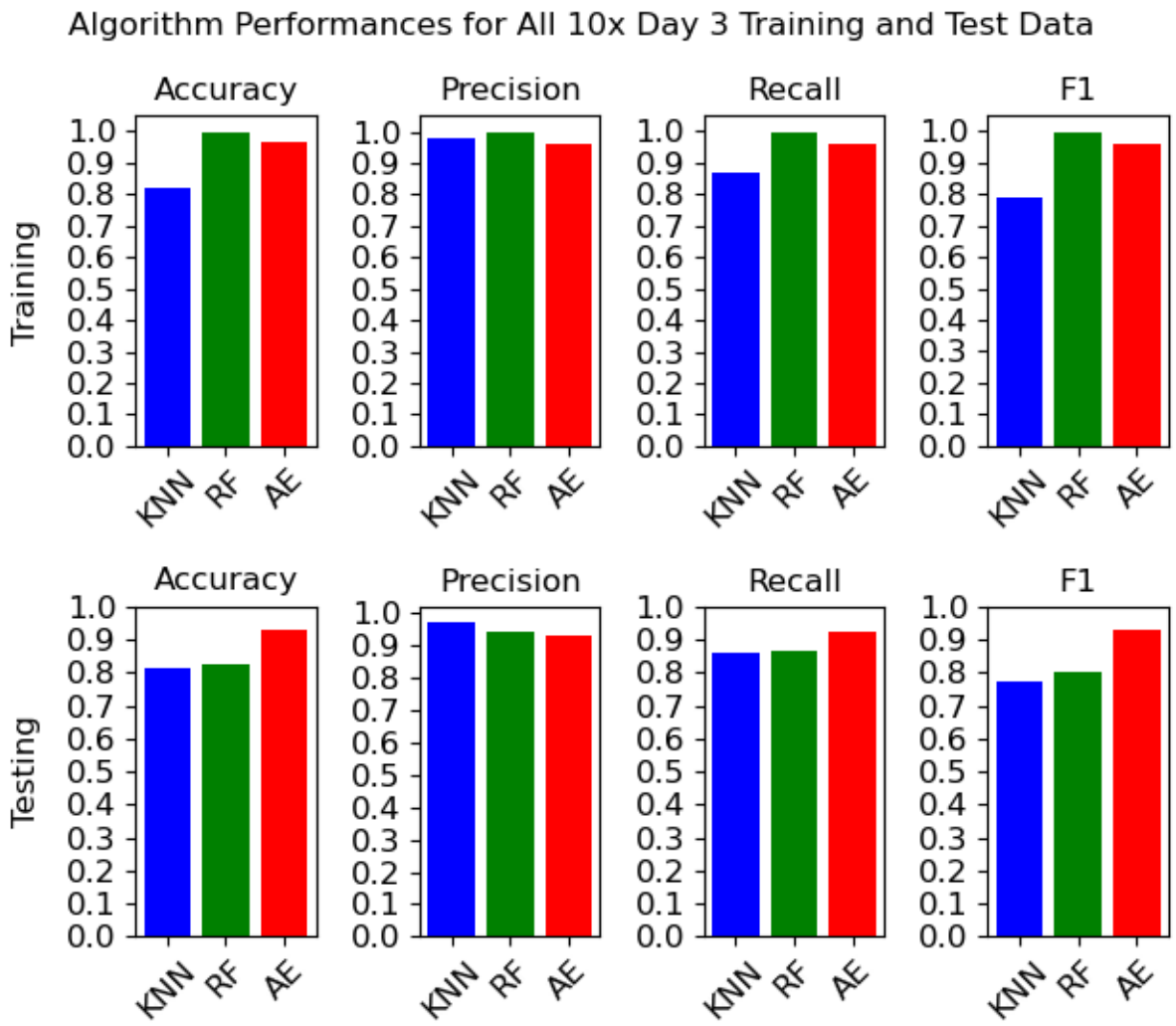


Confusion Matrices for All 10x Day 2 Training and Test Data

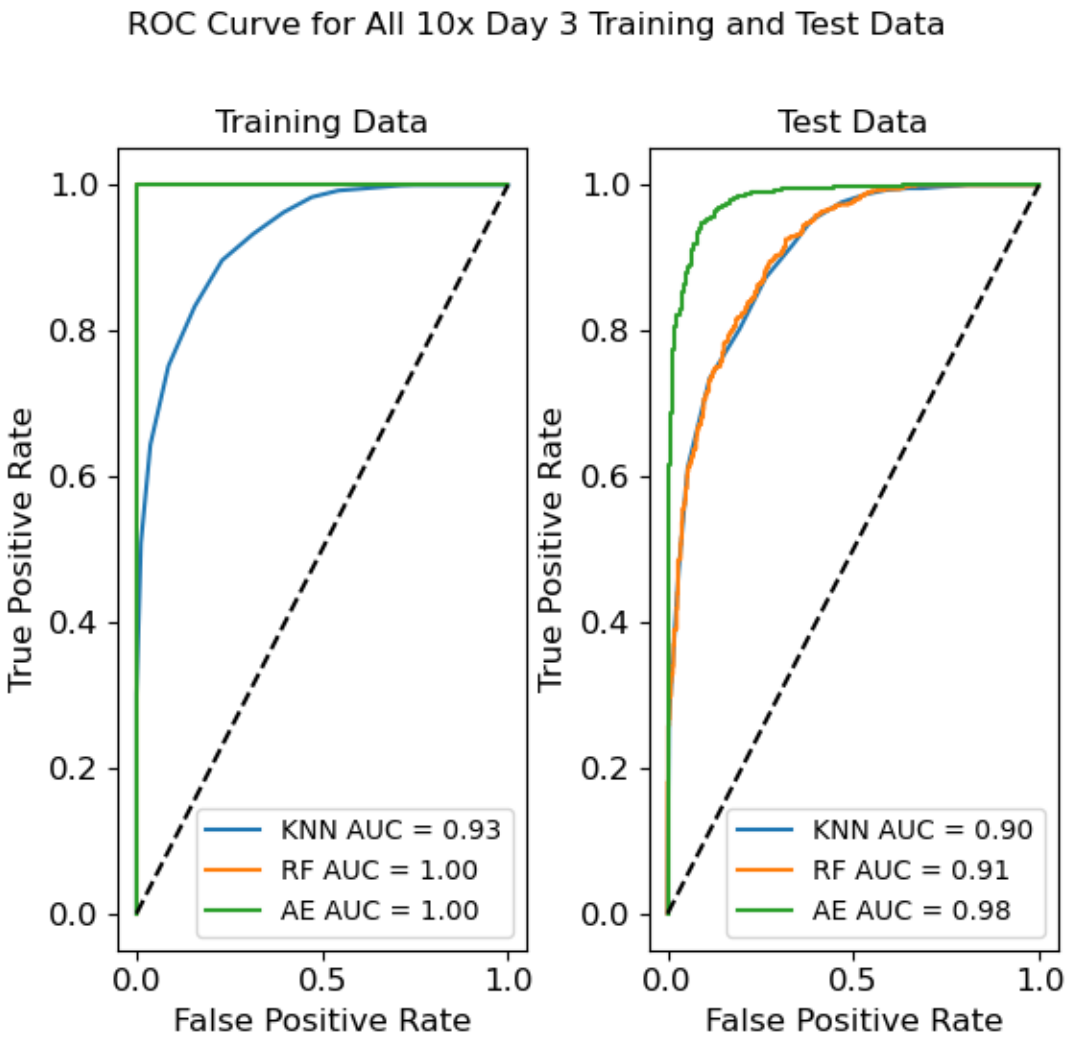


Autoencoder Model Losses for All 10x Day 2 Train and Test Data

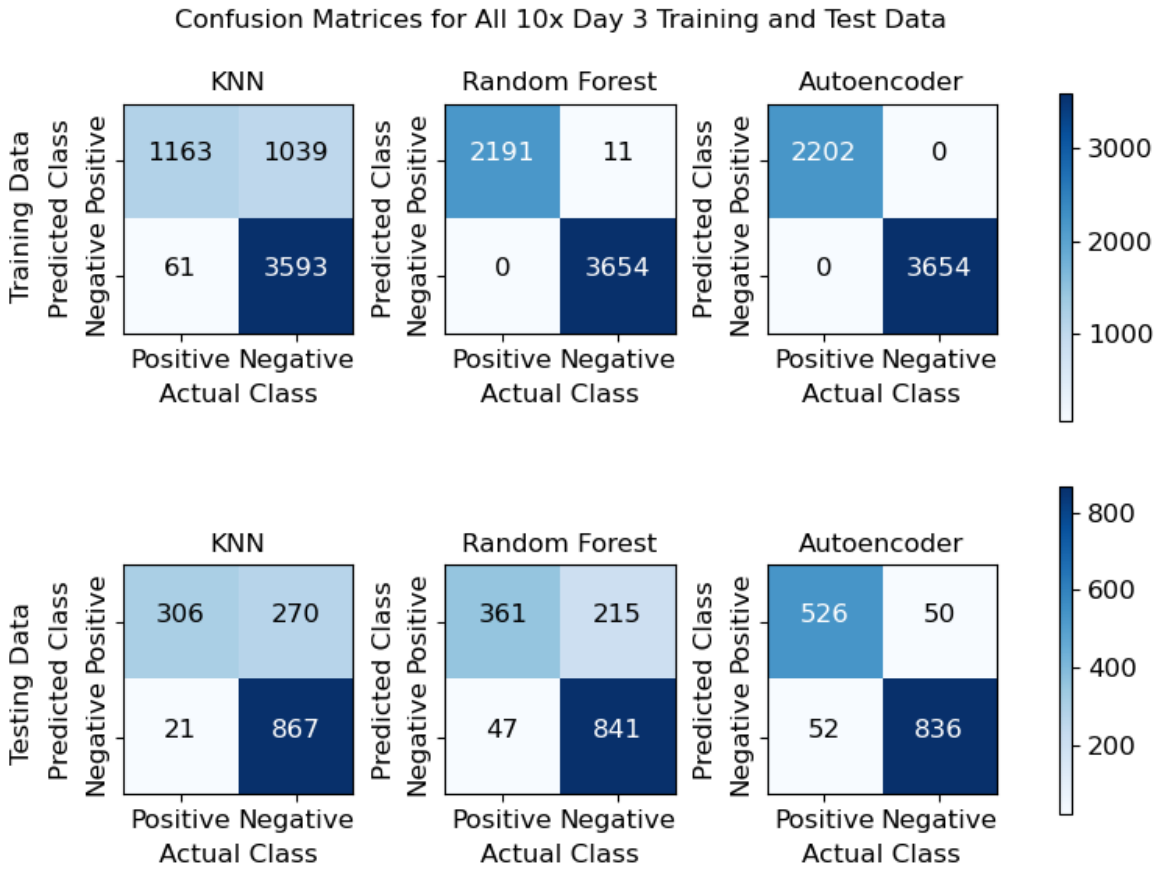
Aggregate Data Evaluation 10x Day 3



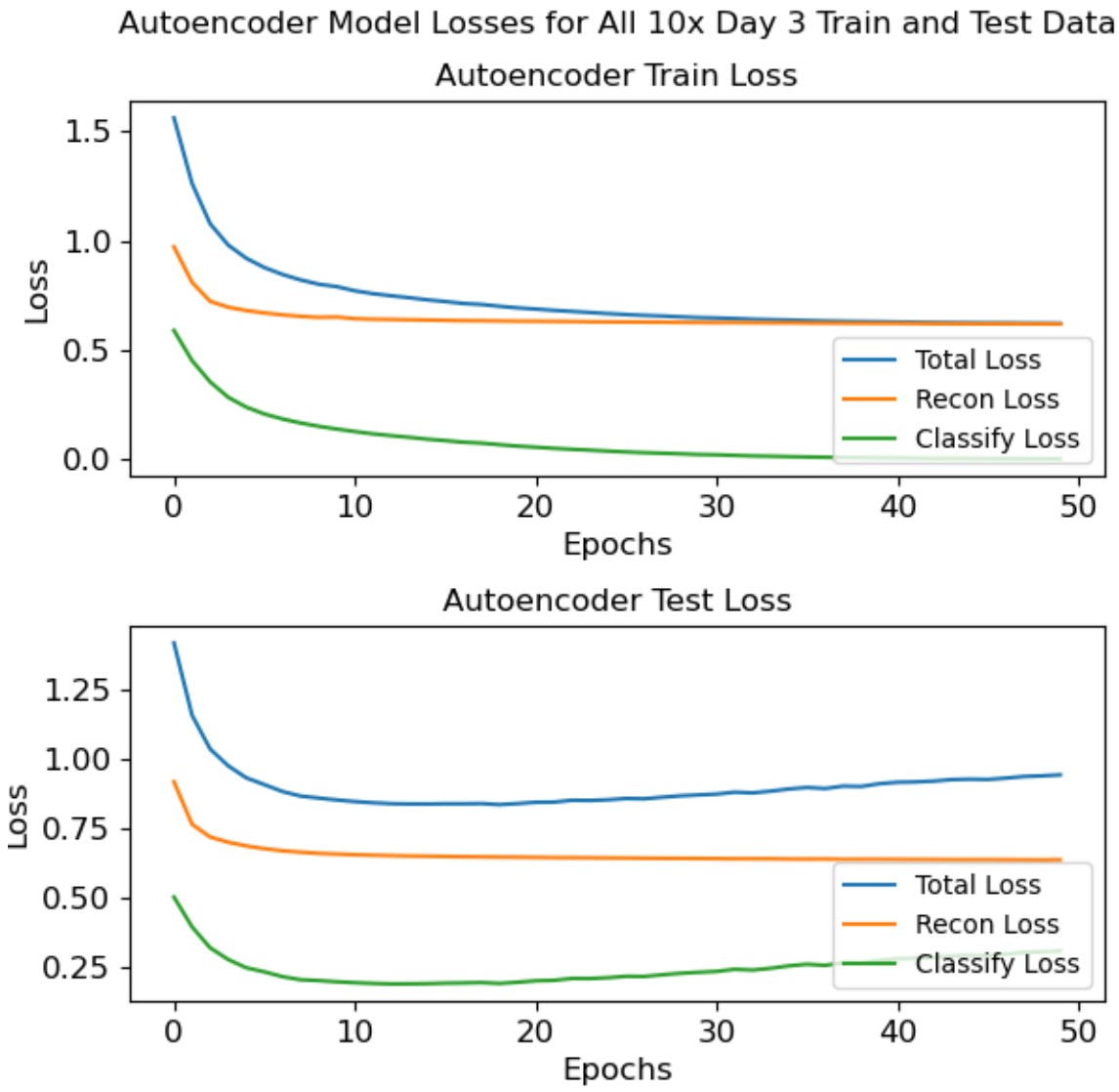
Algorithm Performances for All 10x Day 3 Training and Test Data



ROC Curve for All 10x Day 3 Training and Test Data

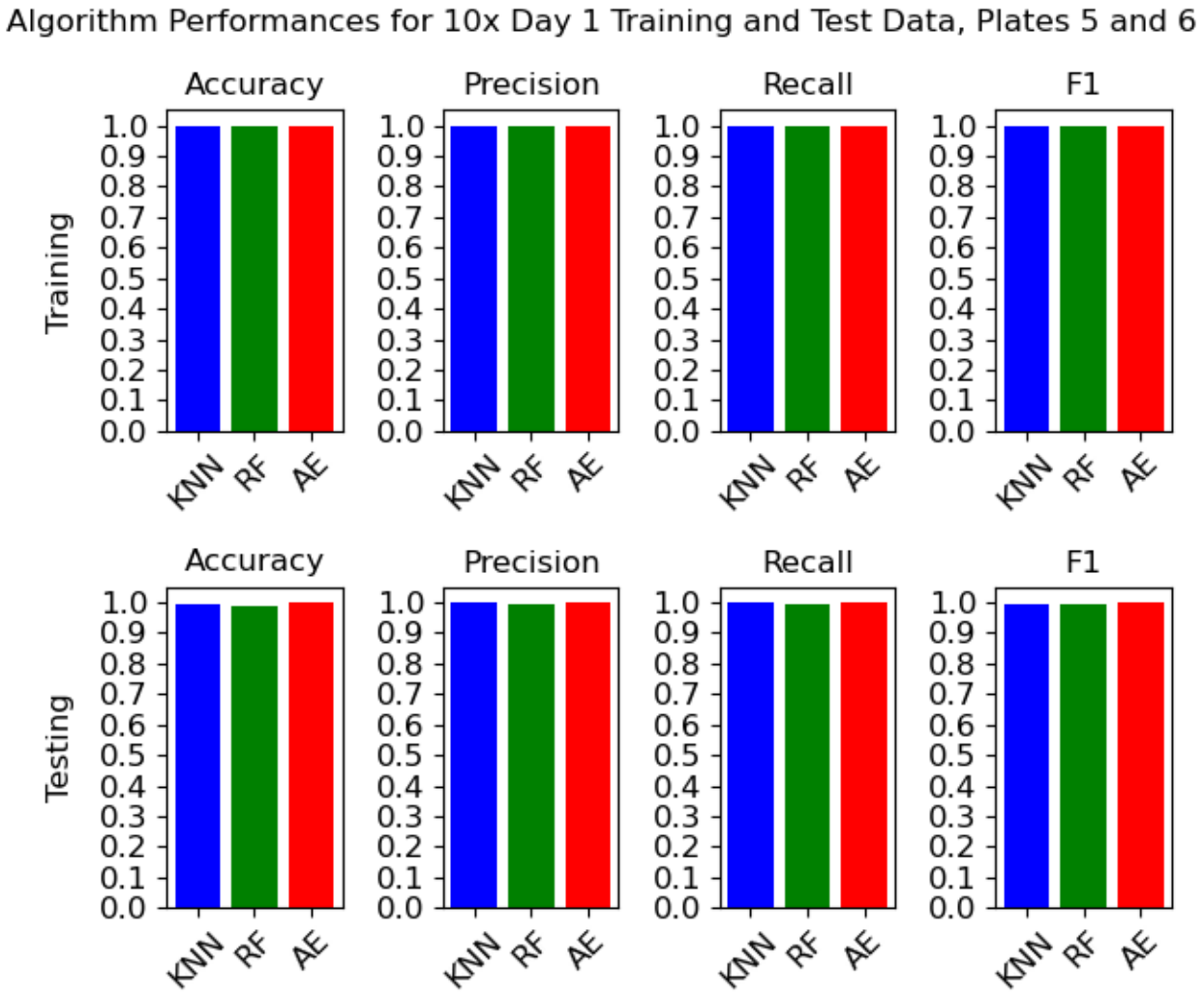


Confusion Matrices for All 10x Day 3 Training and Test Data

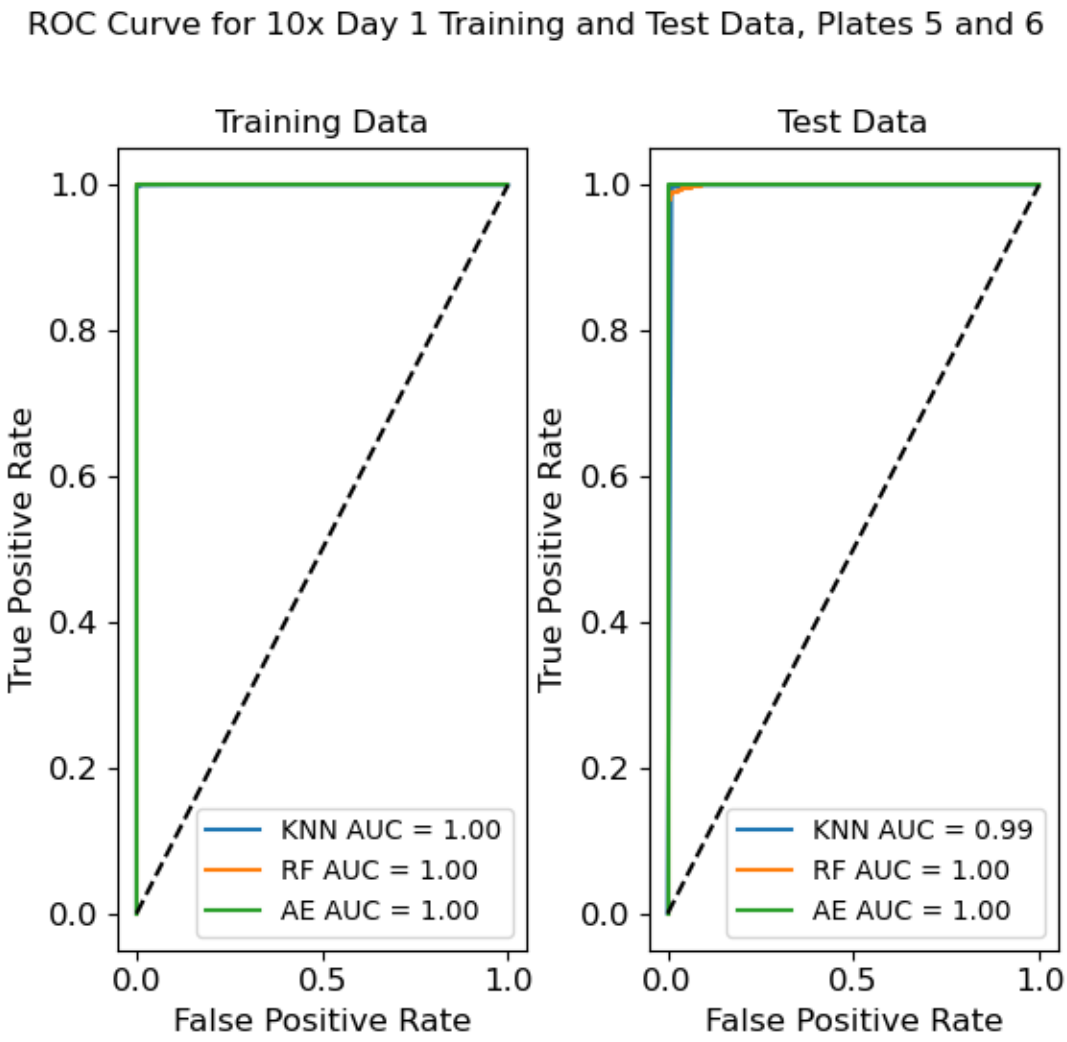


Autoencoder Model Losses for All 10x Day 3 Train and Test Data

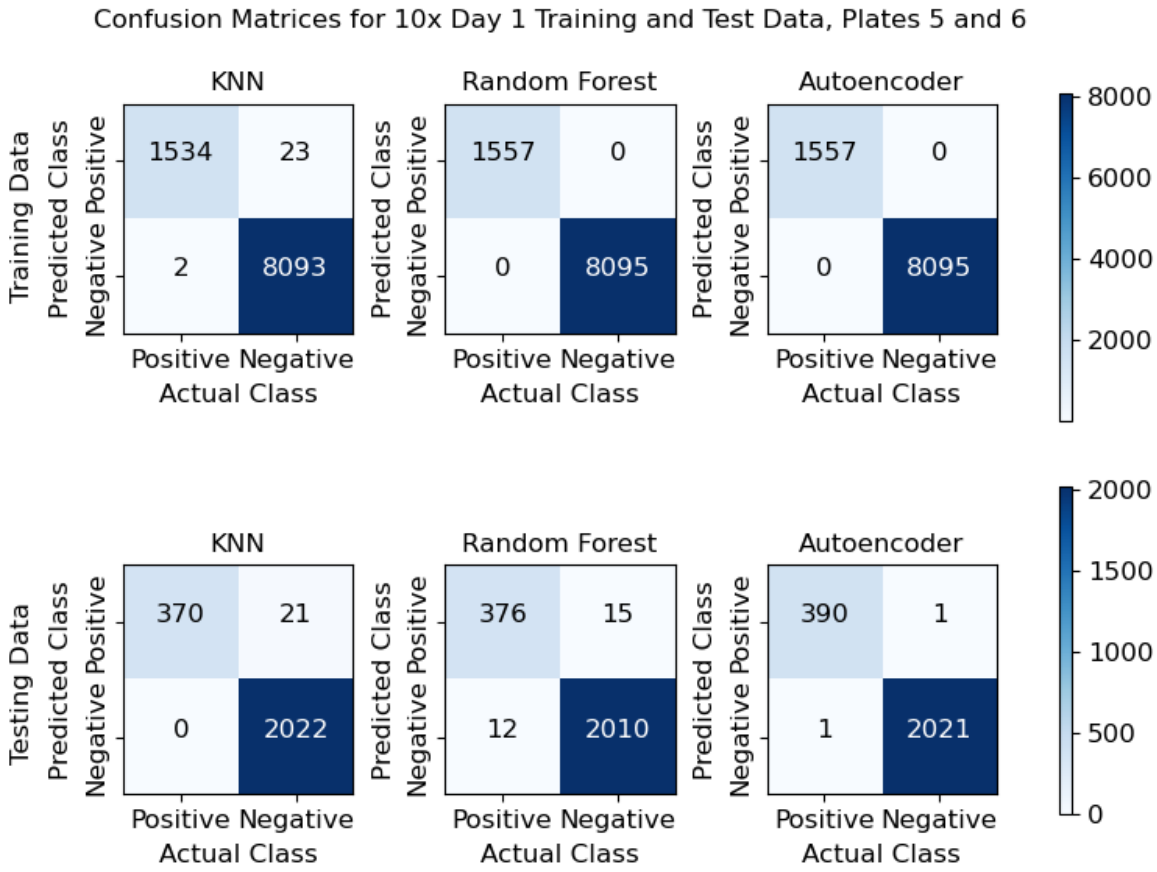
Plates 5 and 6 10x Data Figures



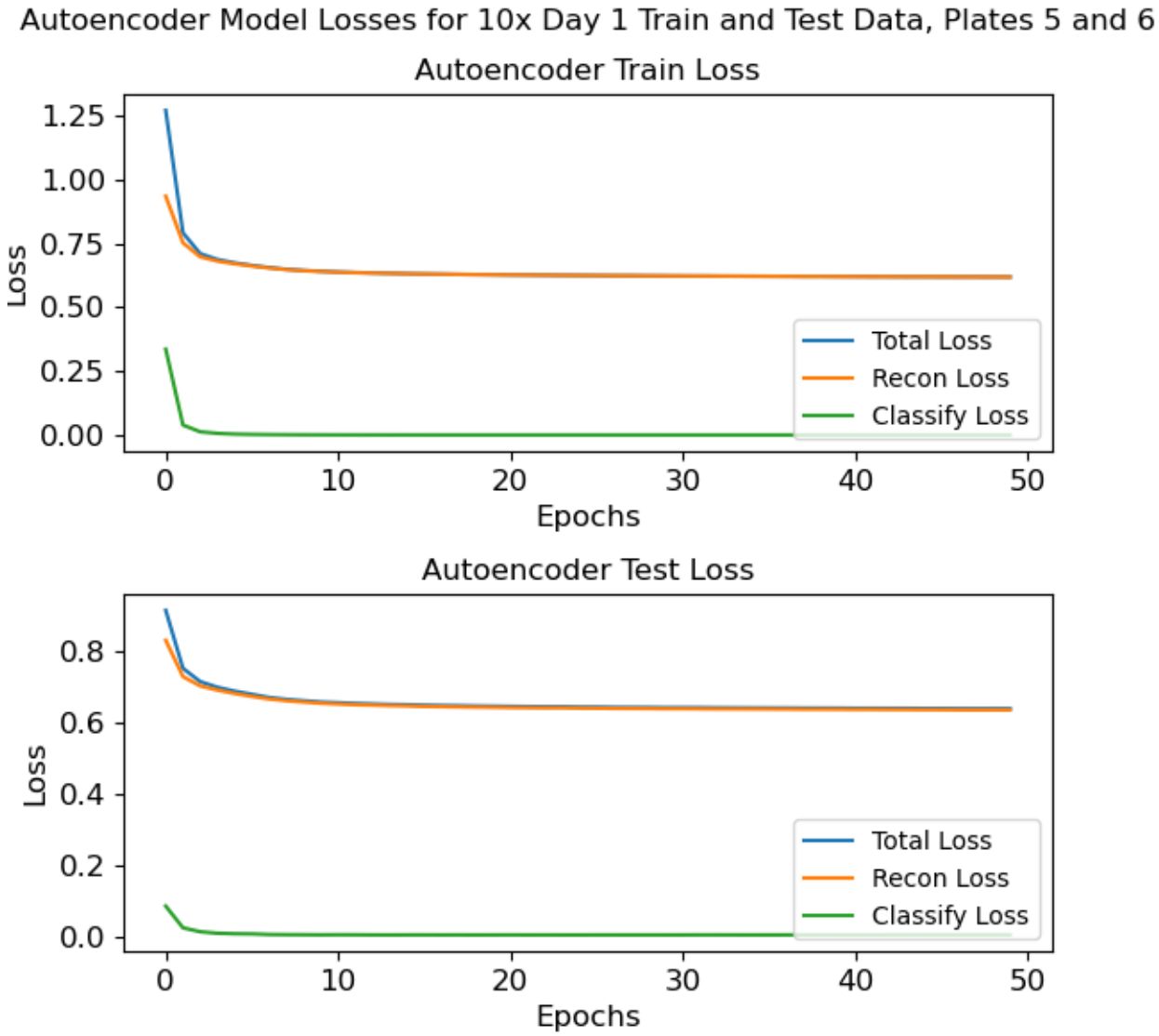
Algorithm Performances for 10x Day 1 Training and Test Data, Plates 5 and 6



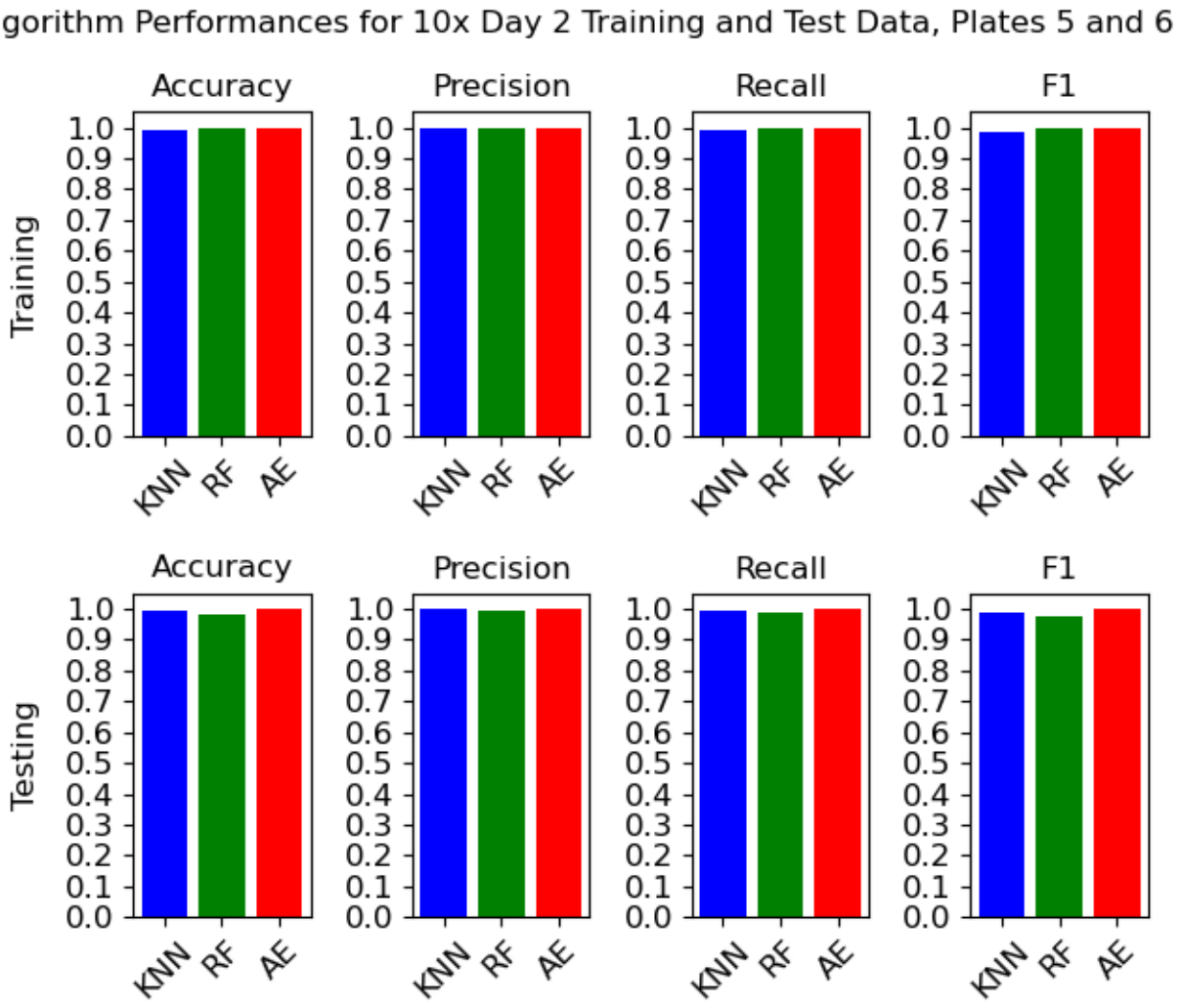
ROC Curve for 10x for Day 1 Training and Test Data, Plates 5 and 6



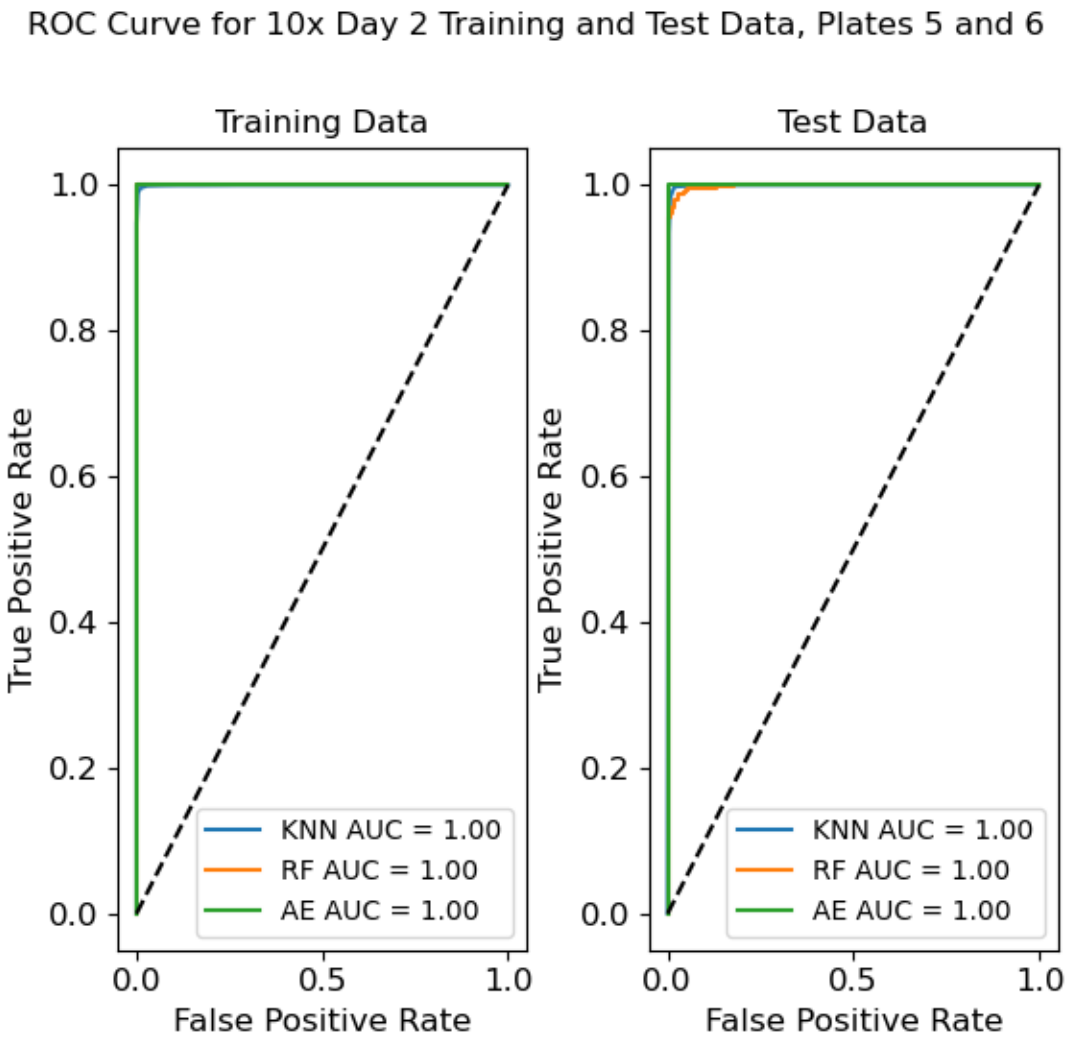
Confusion Matrices for 10x Day 1 Training and Test Data, Plates 5 and 6



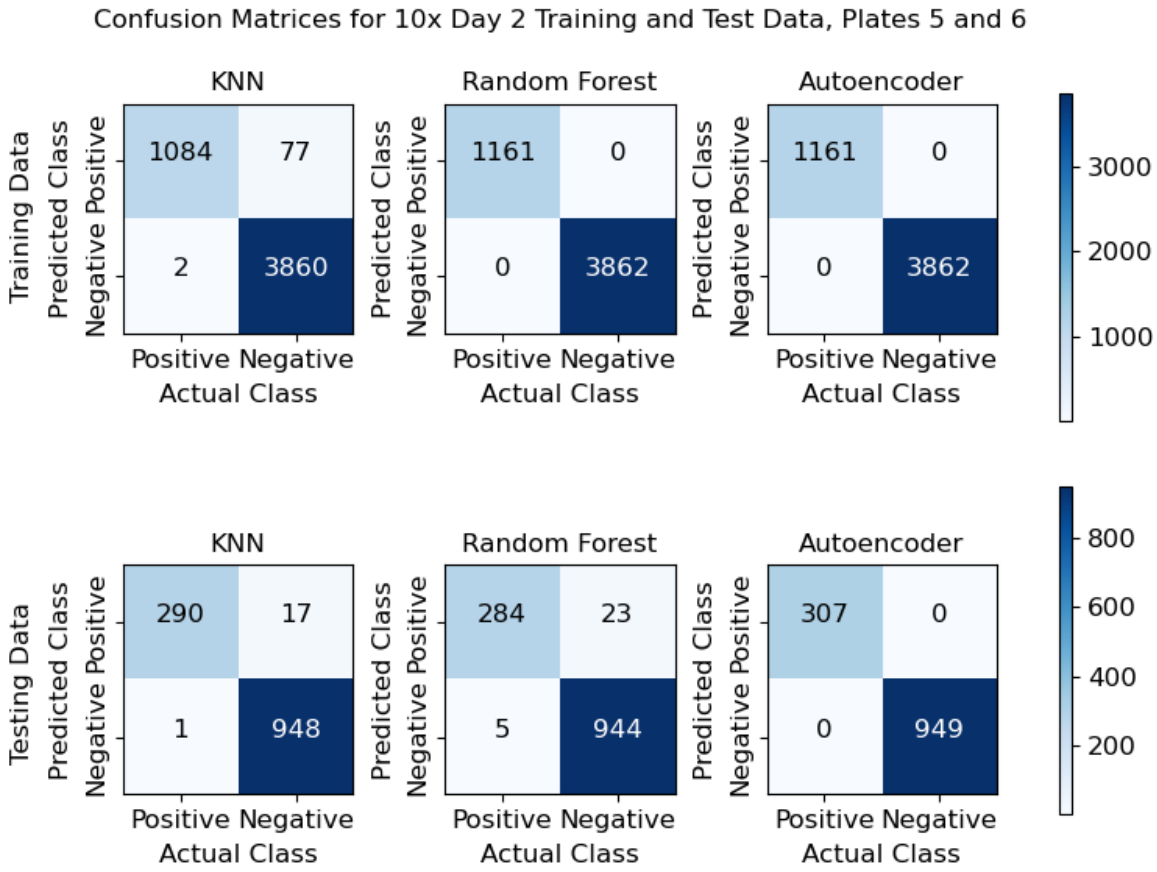
Autoencoder Model Losses for 10x Day 1 Train and Test Data, Plates 5 and 6



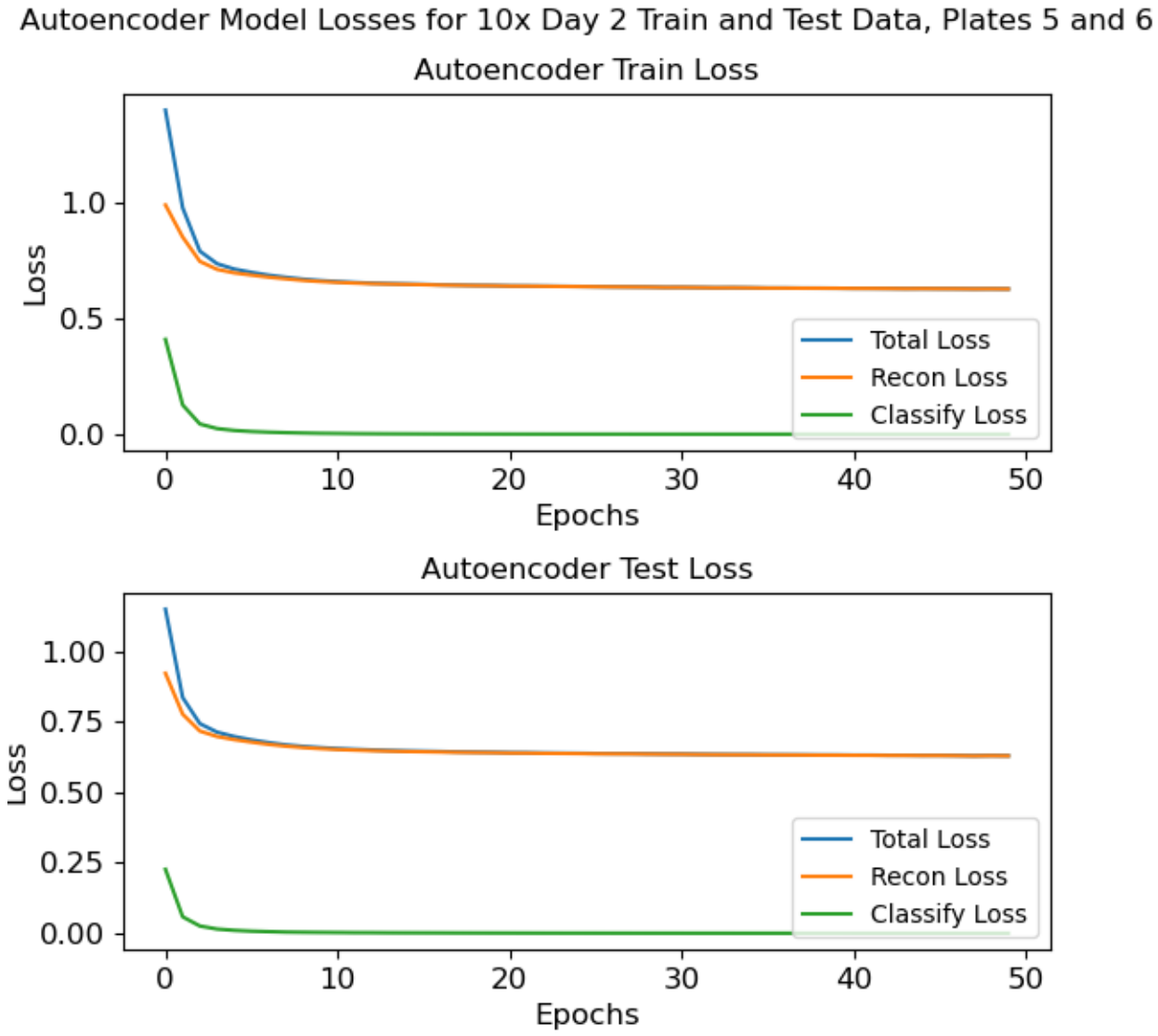
Algorithm Performances for 10x Day 2 Training and Test Data, Plates 5 and 6



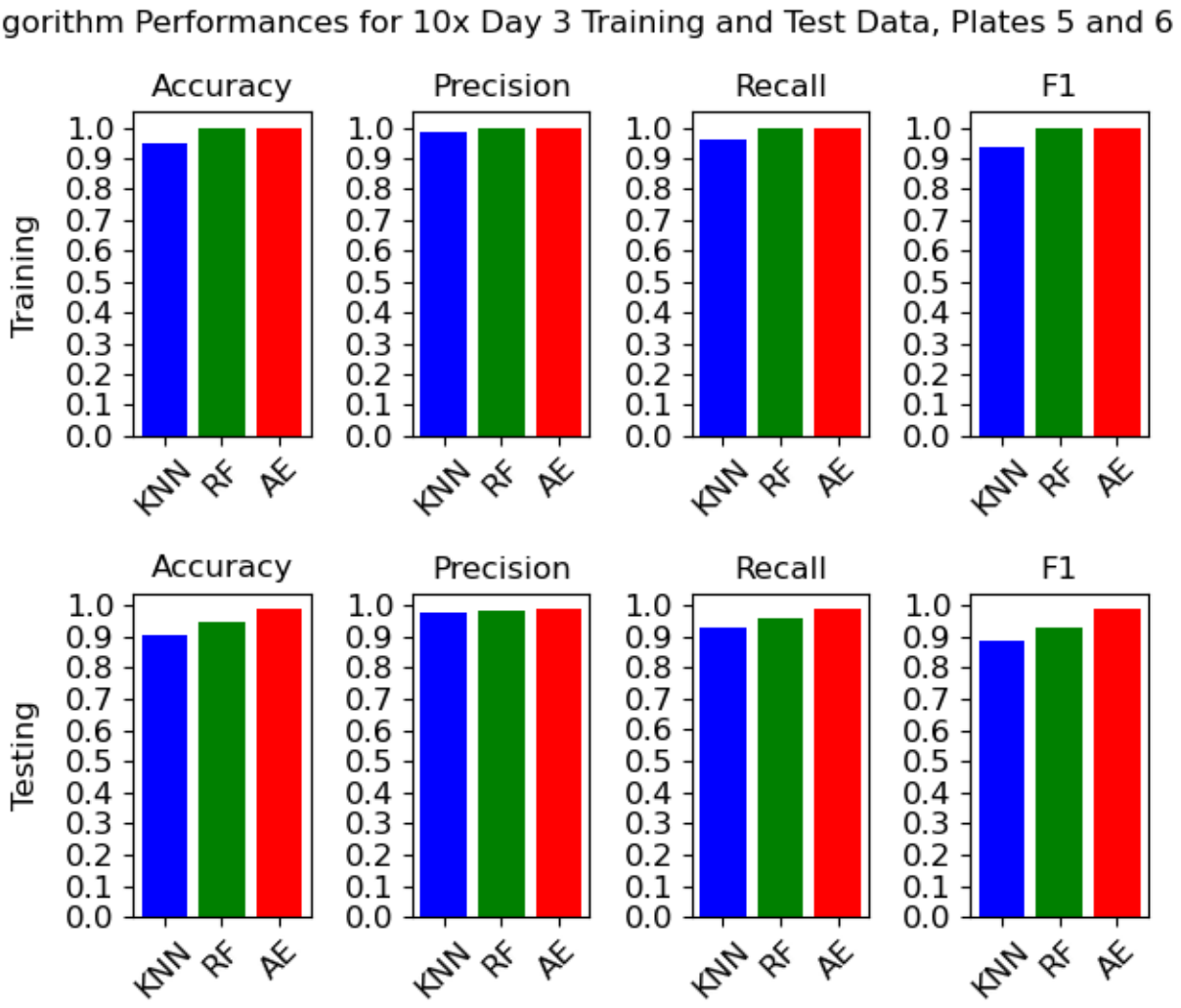
ROC Curve for 10x for Day 2 Training and Test Data, Plates 5 and 6



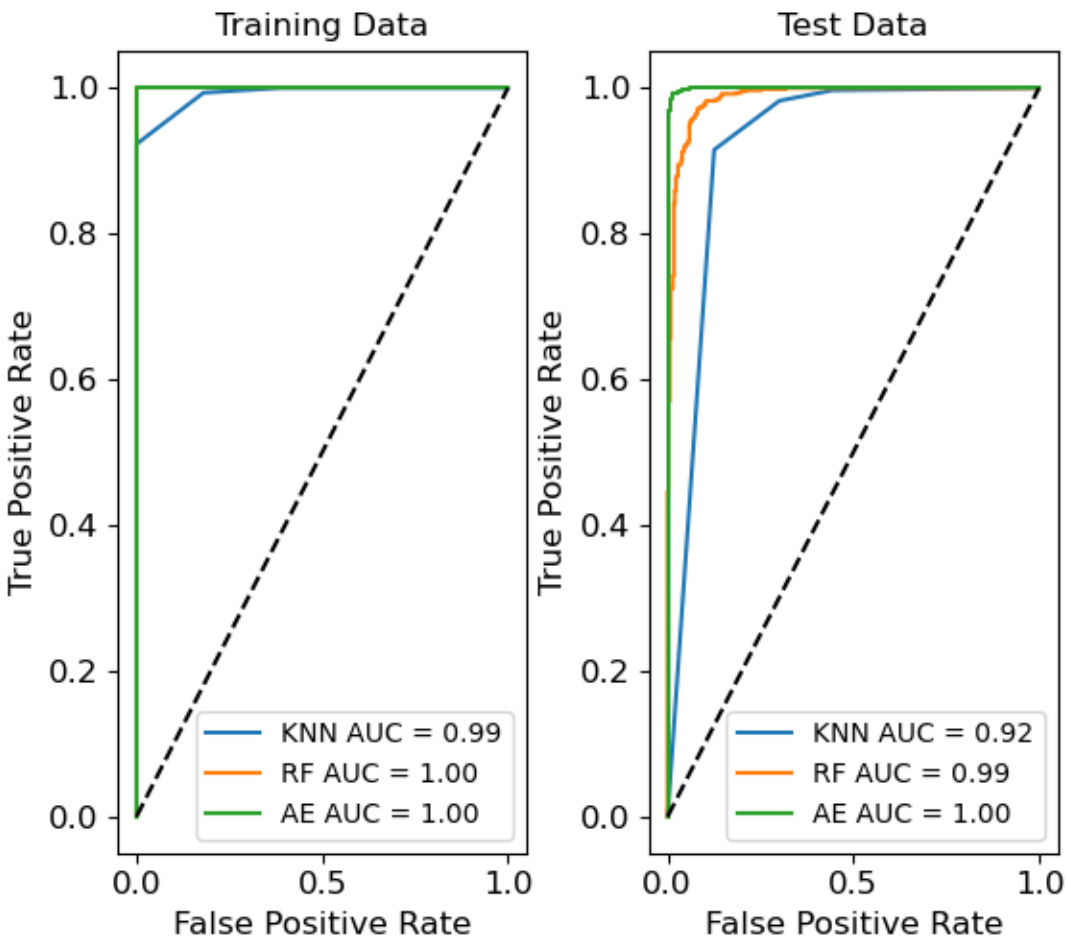
Confusion Matrices for 10x Day 2 Training and Test Data, Plates 5 and 6



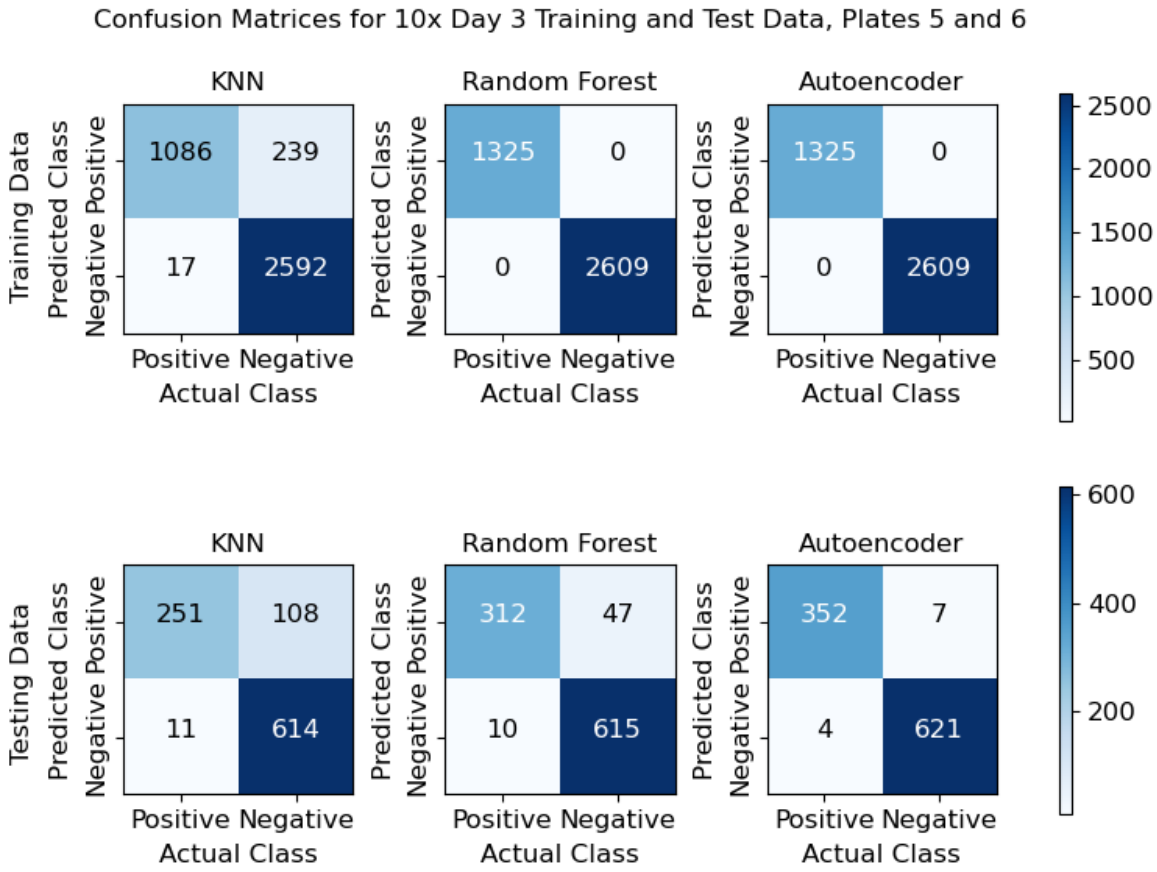
Autoencoder Model Losses for 10x Day 2 Train and Test Data, Plates 5 and 6



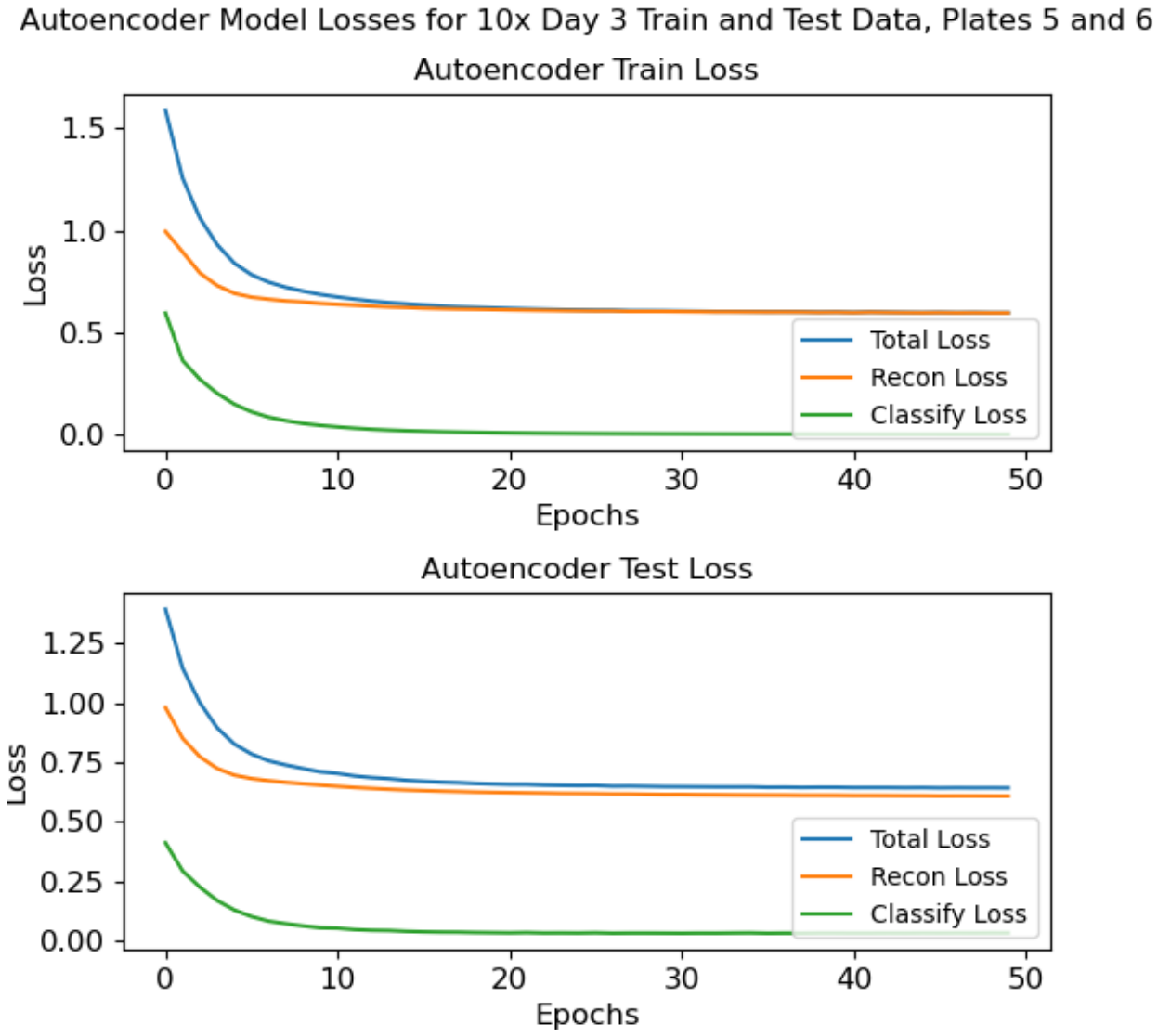
Algorithm Performances for 10x Day 3 Training and Test Data, Plates 5 and 6



ROC Curve for 10x for Day 3 Training and Test Data, Plates 5 and 6

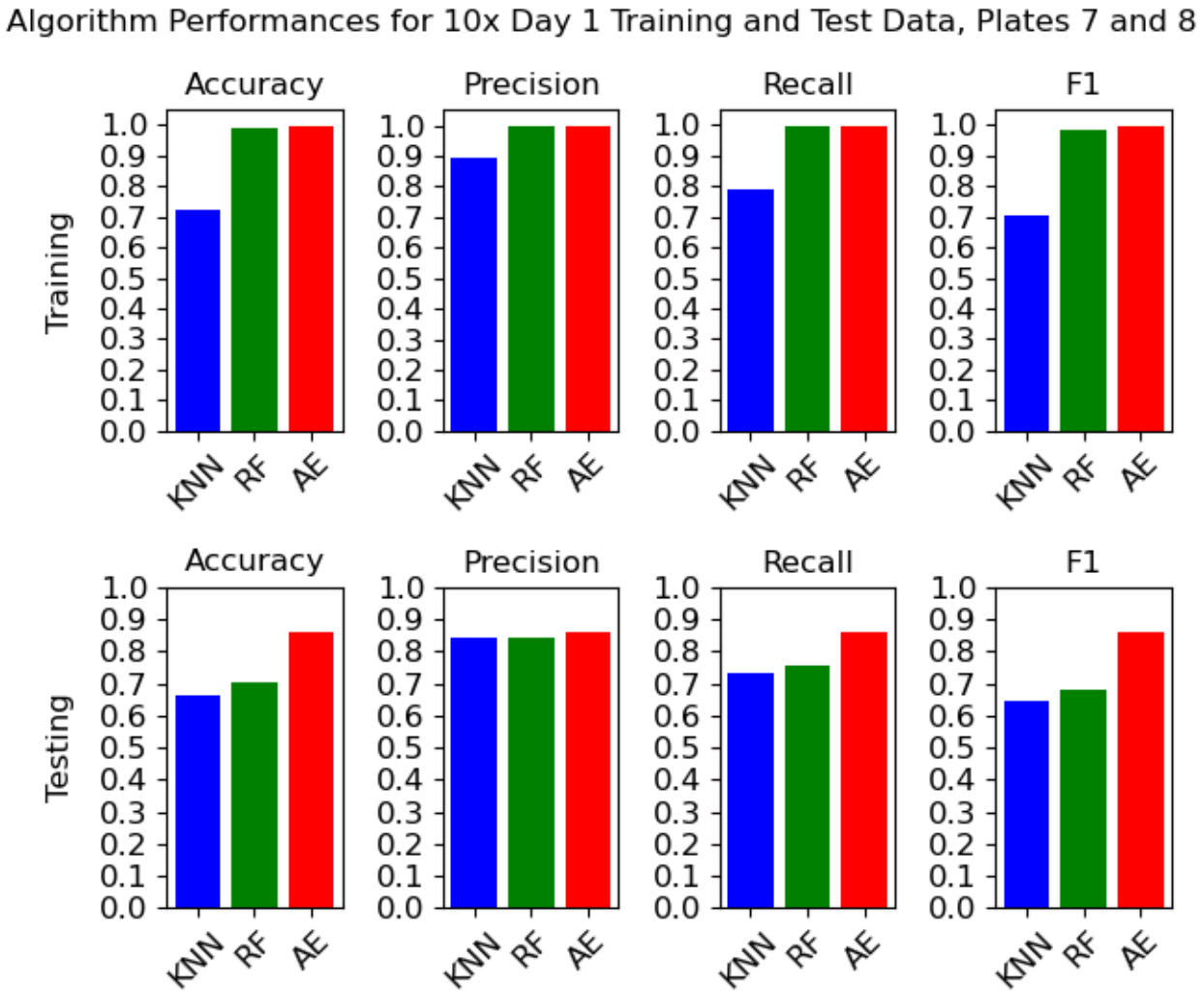


Confusion Matrices for 10x Day 3 Training and Test Data, Plates 5 and 6

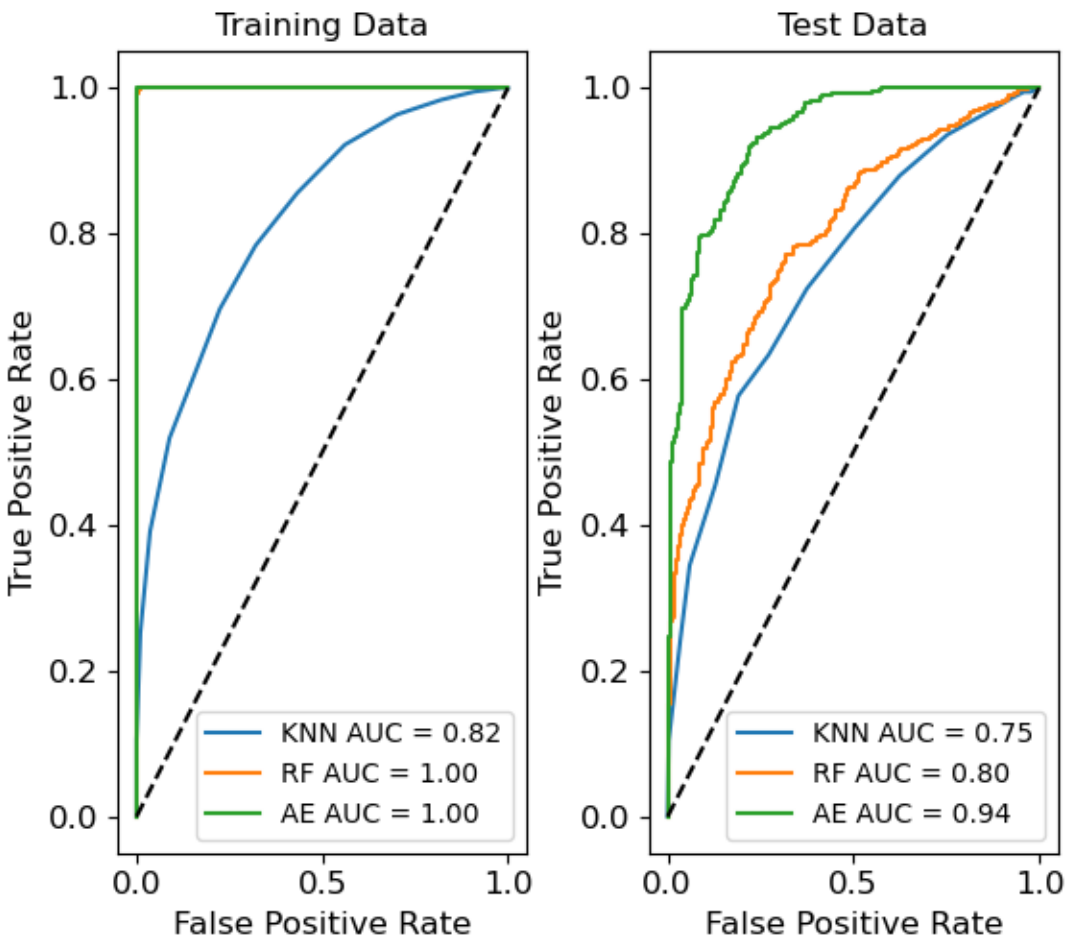


Autoencoder Model Losses for 10x Day 3 Train and Test Data, Plates 5 and 6

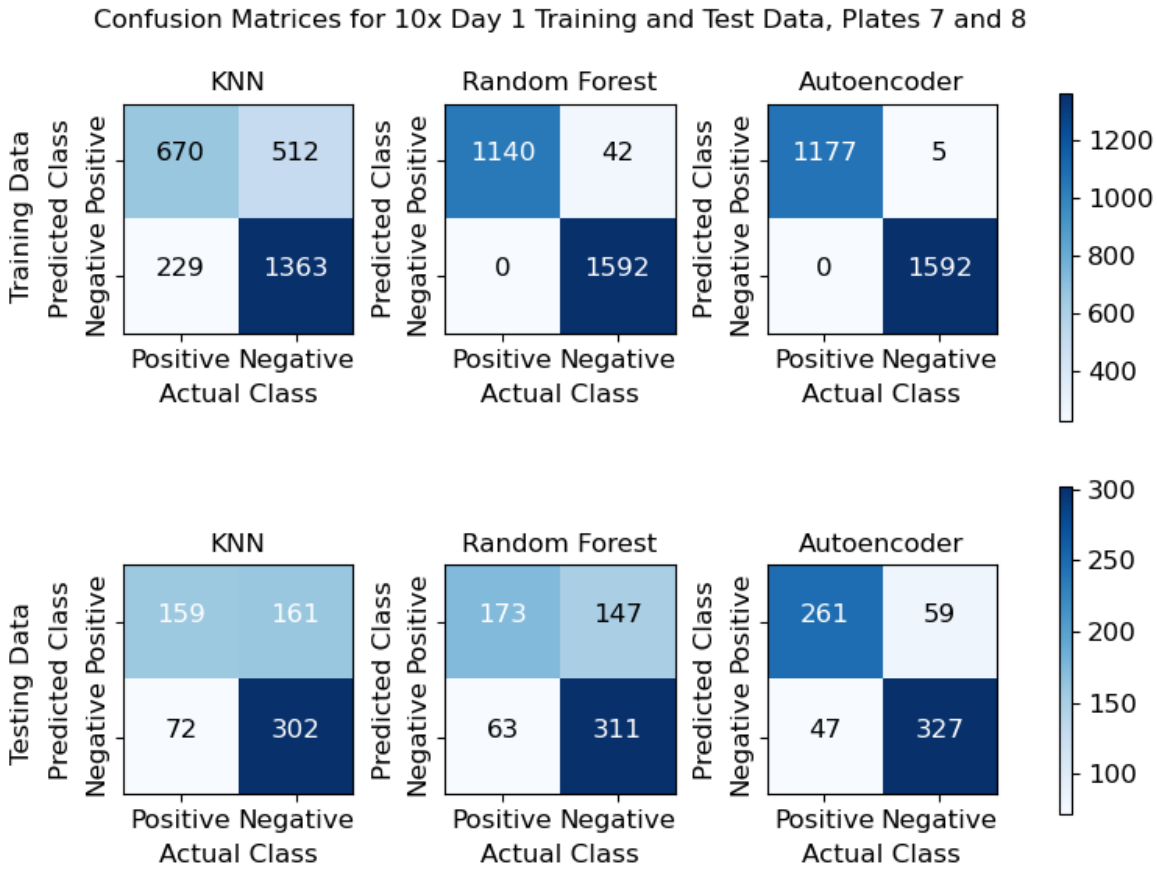
Plates 7 and 8 10x Data Figures



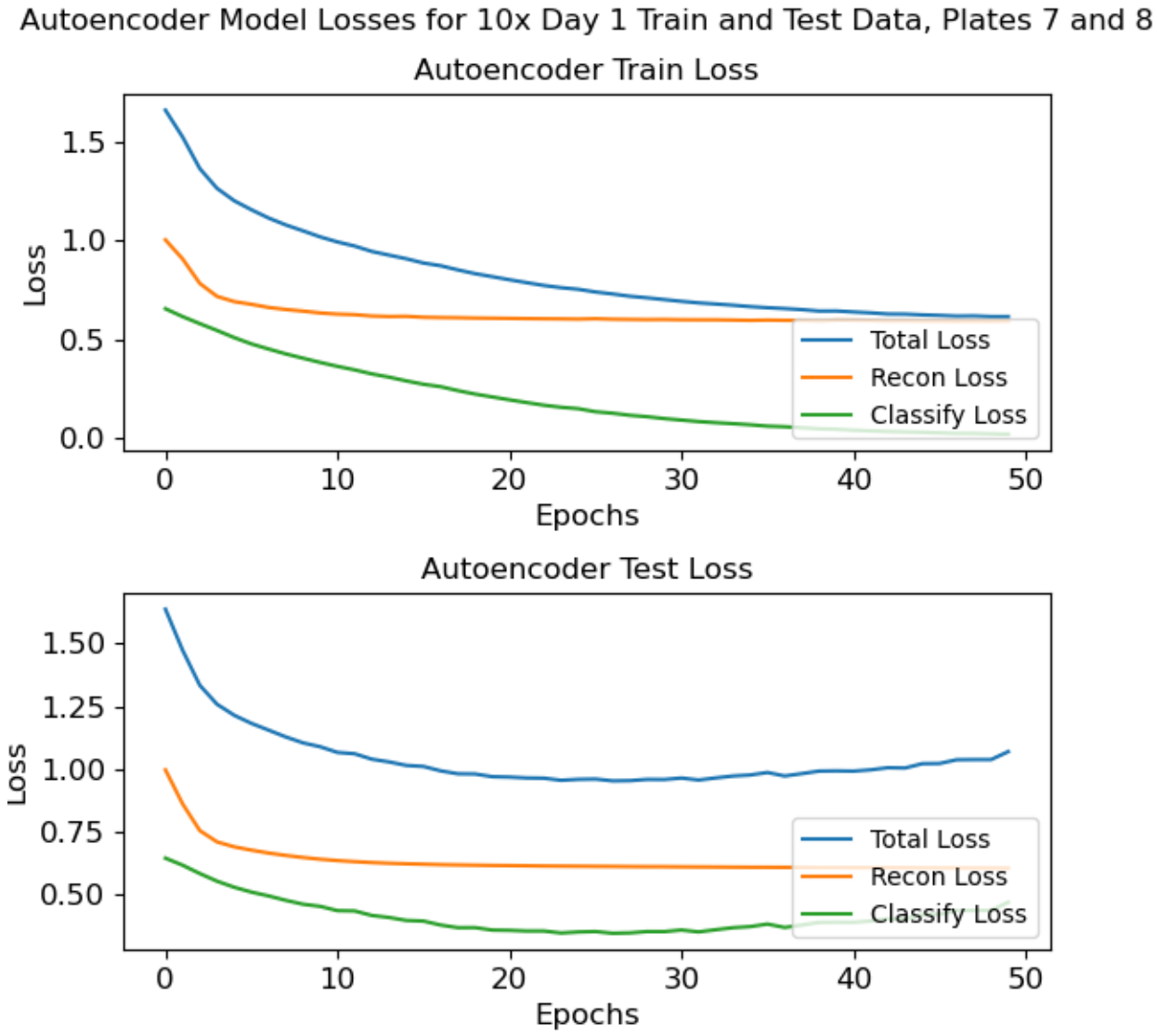
Algorithm Performances for 10x Day 1 Training and Test Data, Plates 7 and 8



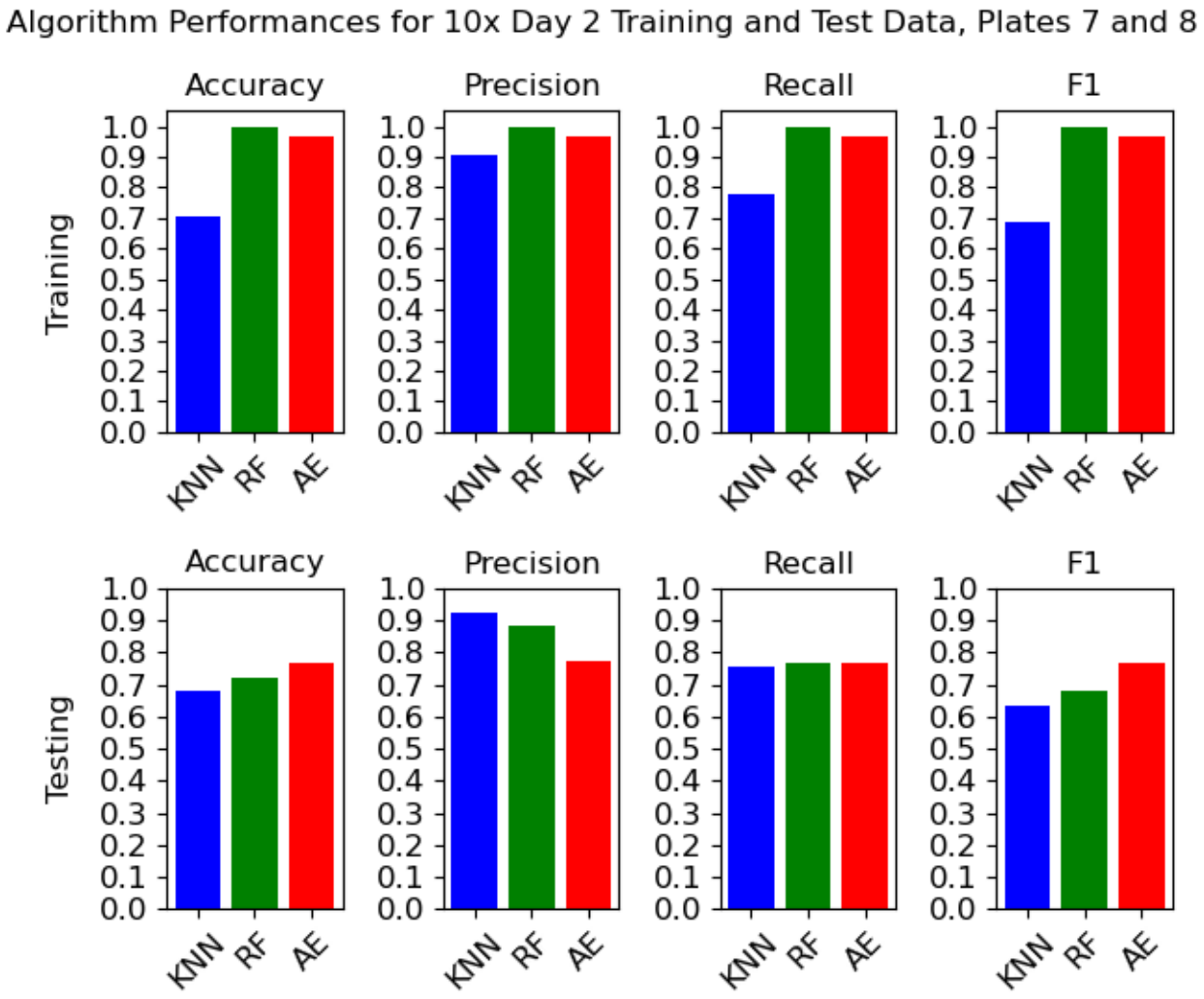
ROC Curve for 10x for Day 1 Training and Test Data, Plates 7 and 8



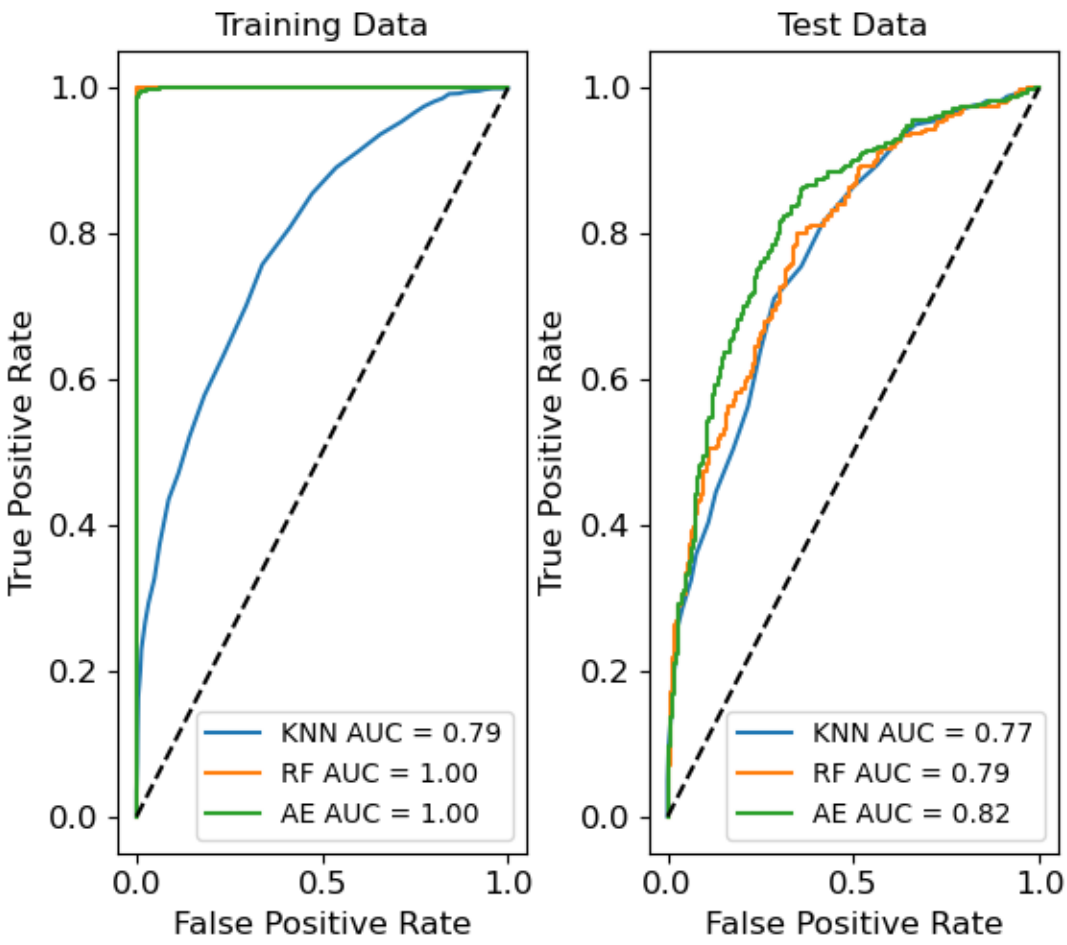
Confusion Matrices for 10x Day 1 Training and Test Data, Plates 7 and 8



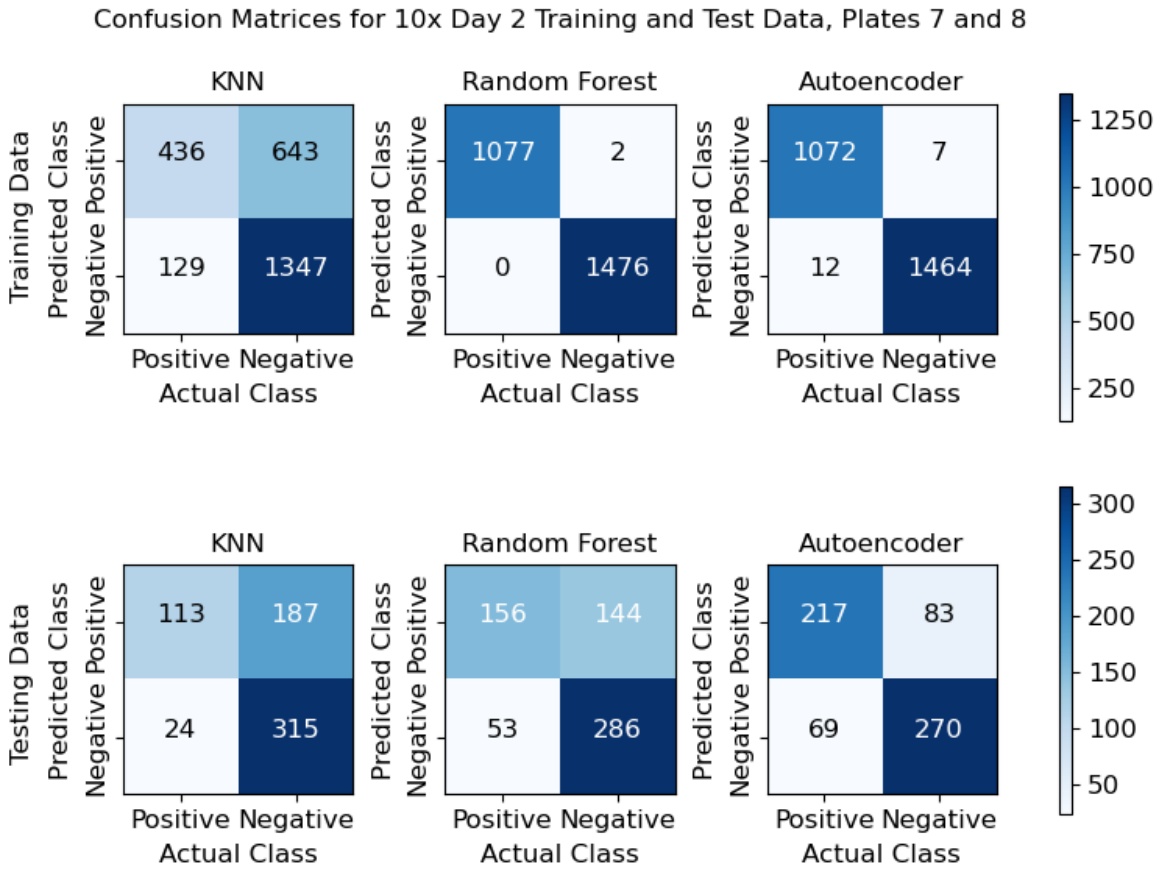
Autoencoder Model Losses for 10x Day 1 Train and Test Data, Plates 7 and 8



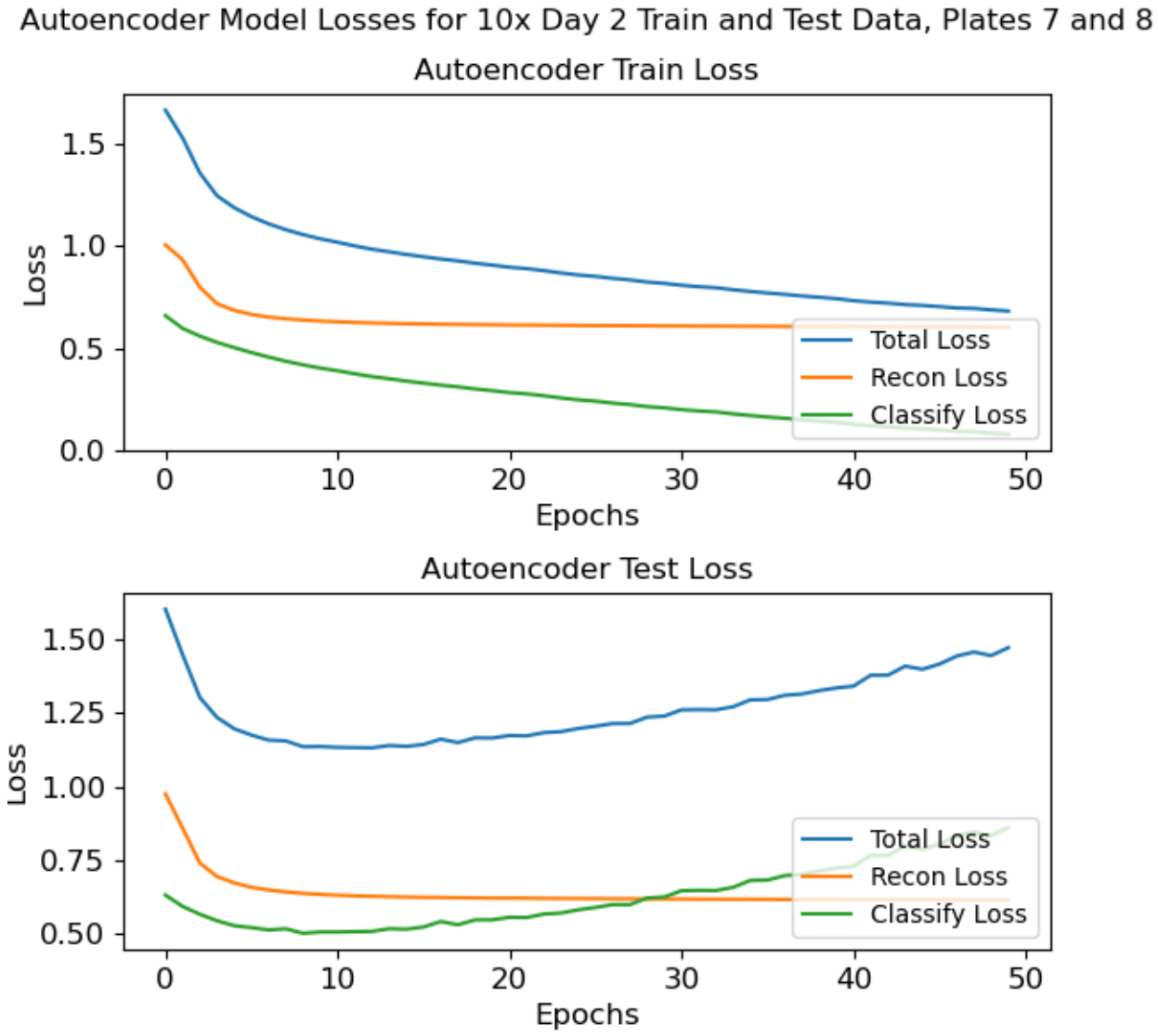
Algorithm Performances for 10x Day 2 Training and Test Data, Plates 7 and 8



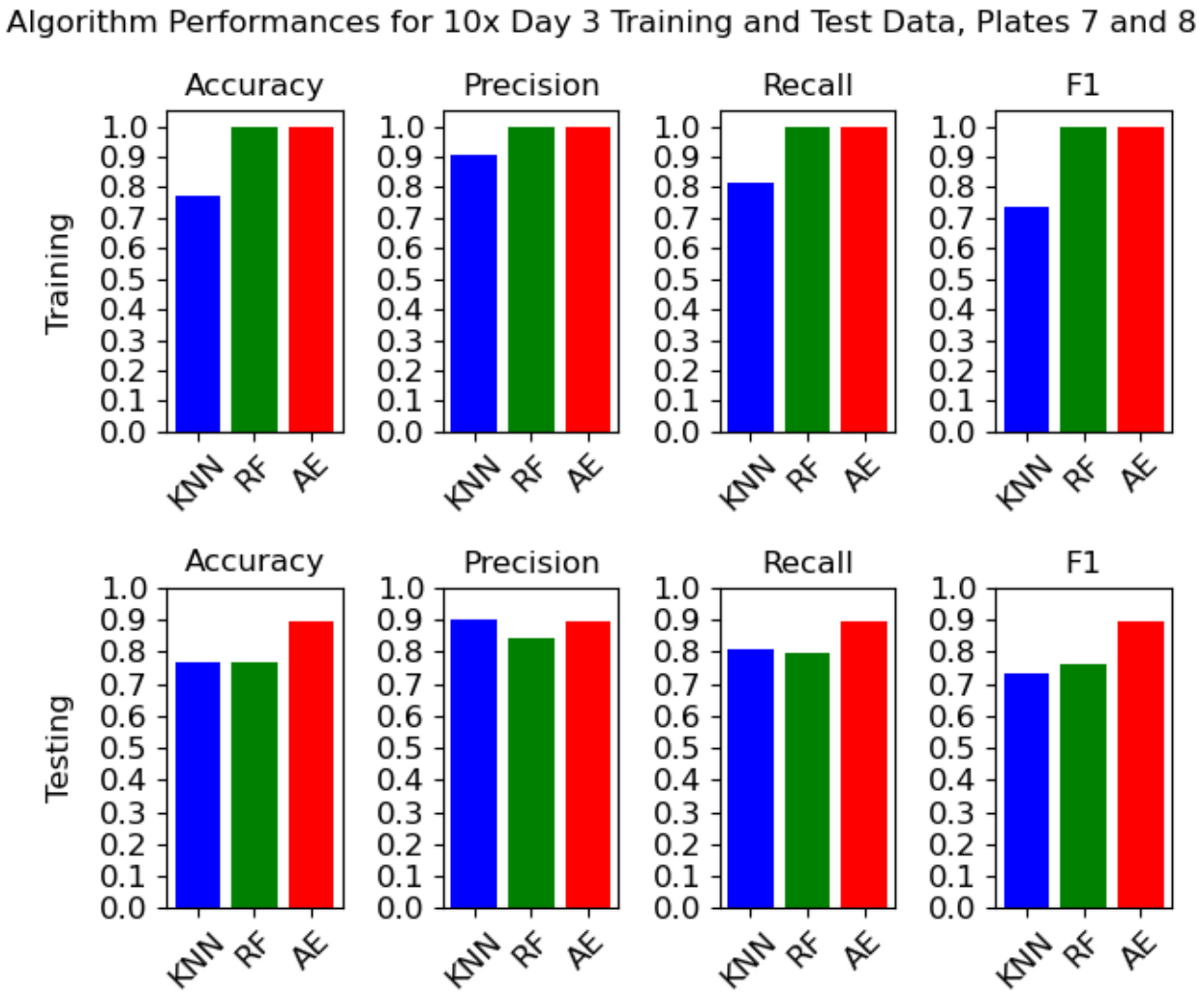
ROC Curve for 10x for Day 2 Training and Test Data, Plates 7 and 8



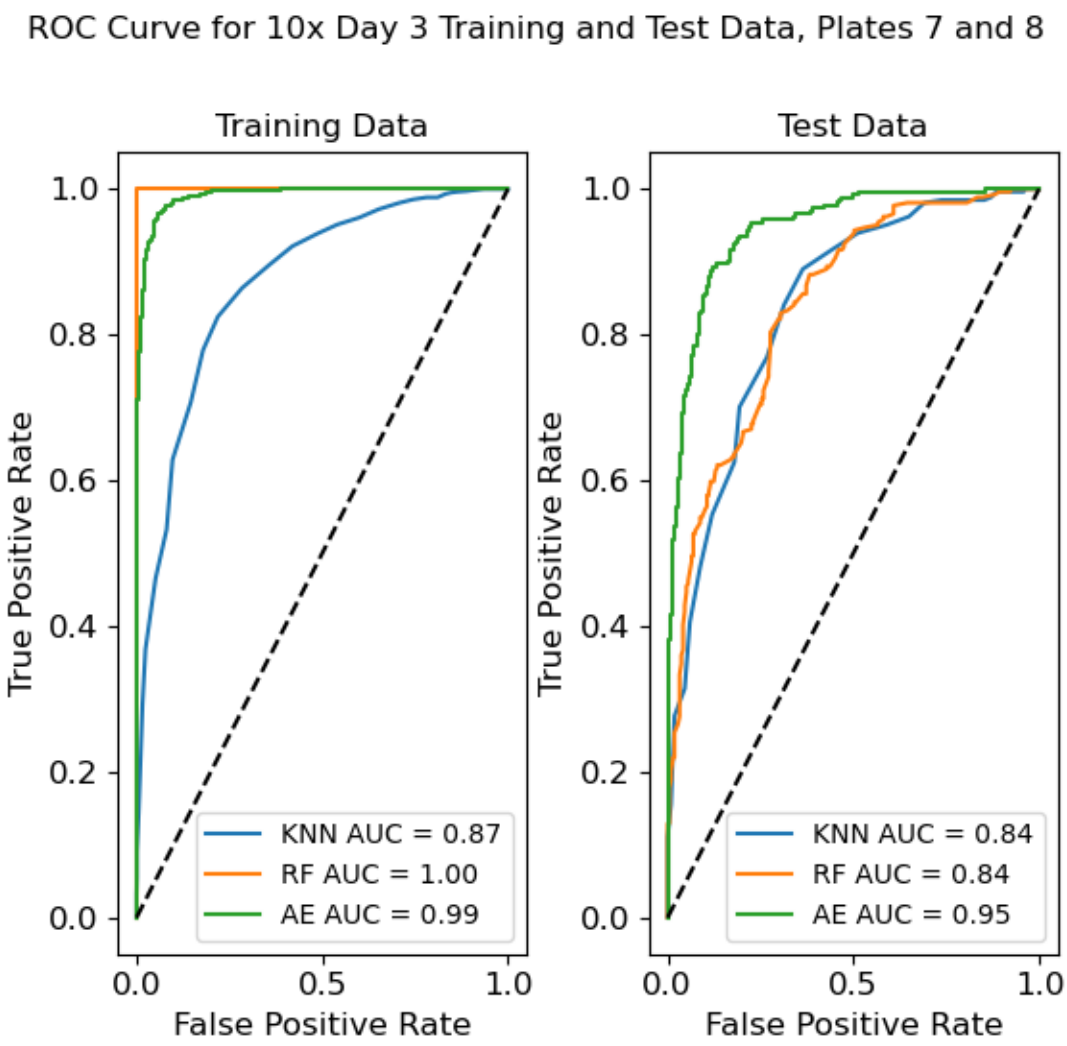
Confusion Matrices for 10x Day 2 Training and Test Data, Plates 7 and 8



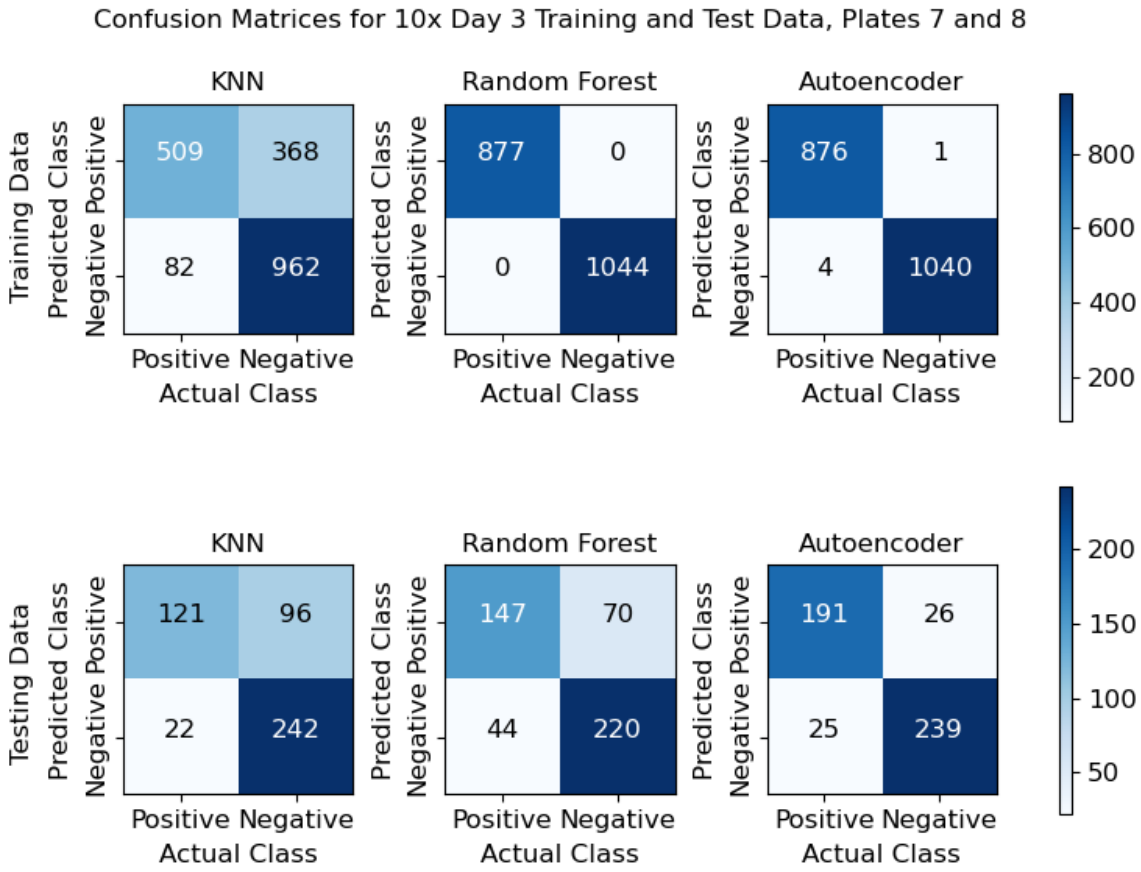
Autoencoder Model Losses for 10x Day 2 Train and Test Data, Plates 7 and 8



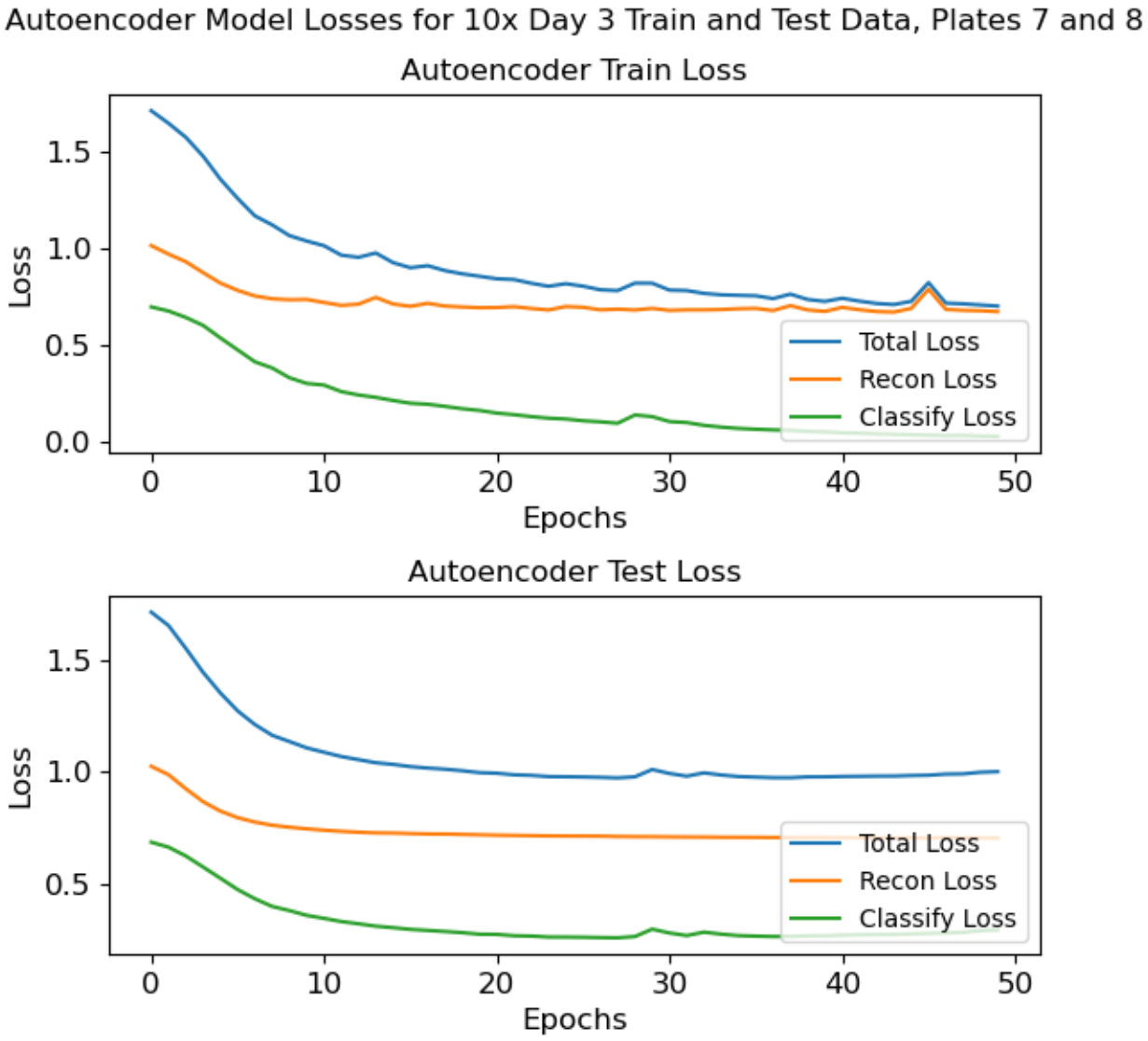
Algorithm Performances for 10x Day 3 Training and Test Data, Plates 7 and 8



ROC Curve for 10x for Day 3 Training and Test Data, Plates 7 and 8

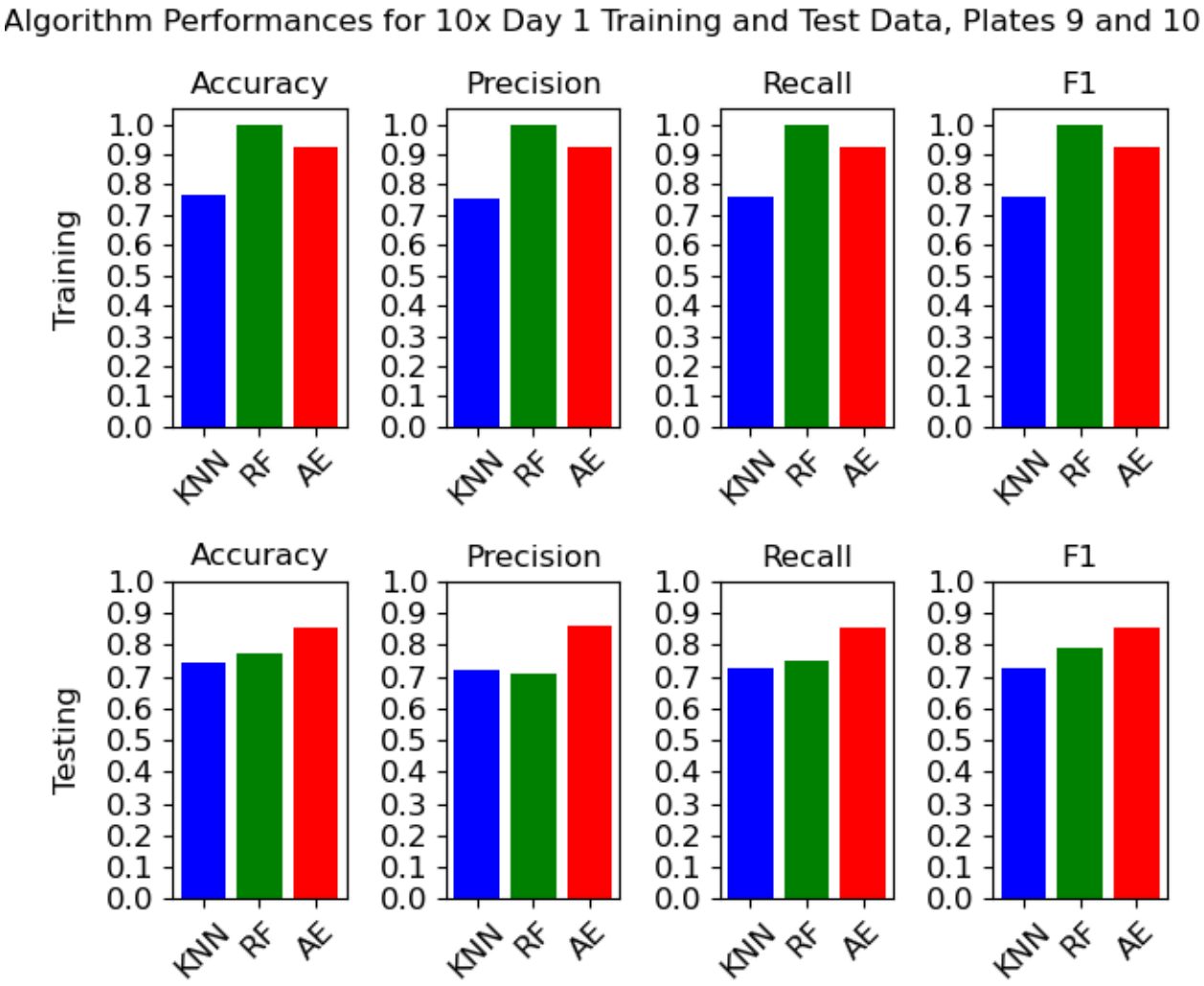


Confusion Matrices for 10x Day 3 Training and Test Data, Plates 7 and 8

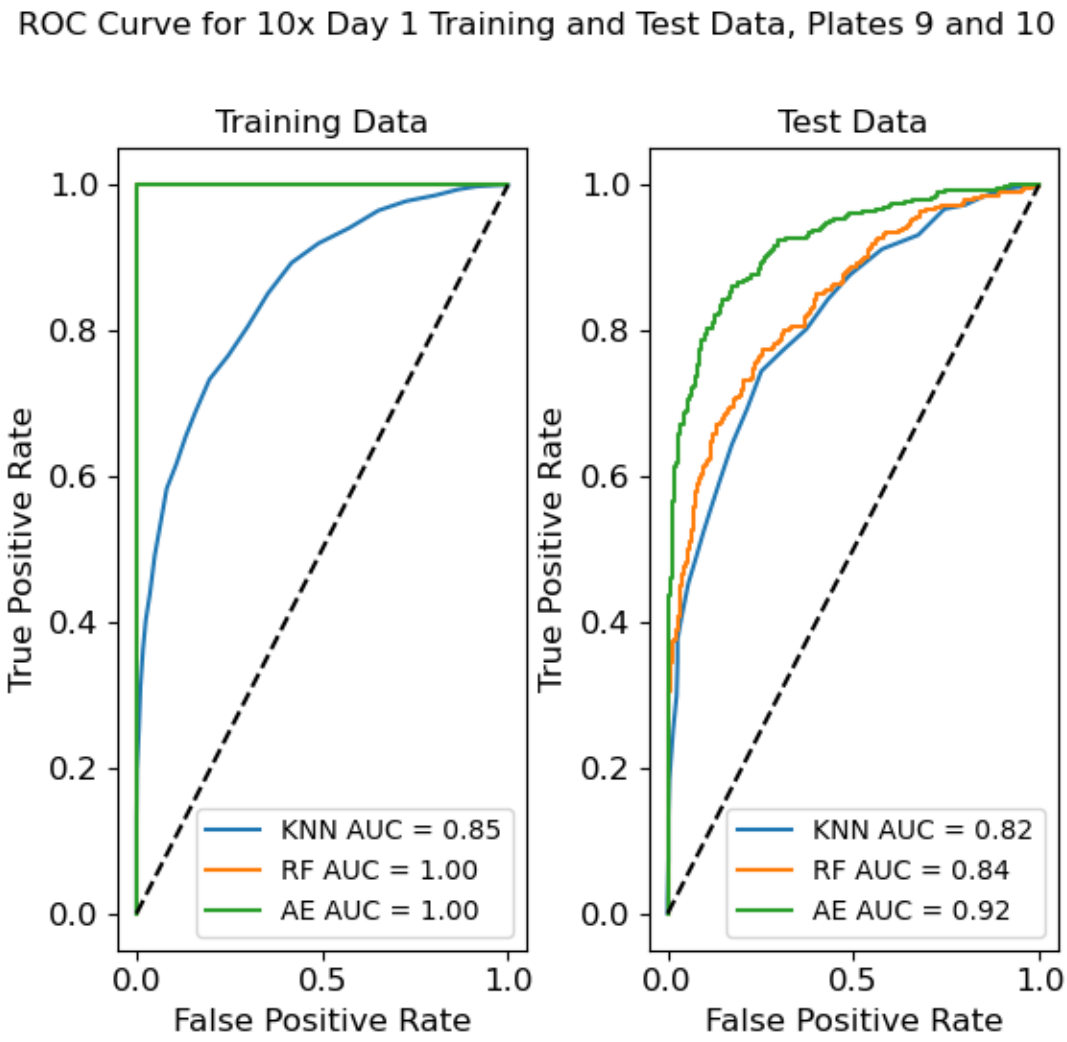


Autoencoder Model Losses for 10x Day 3 Train and Test Data, Plates 7 and 8

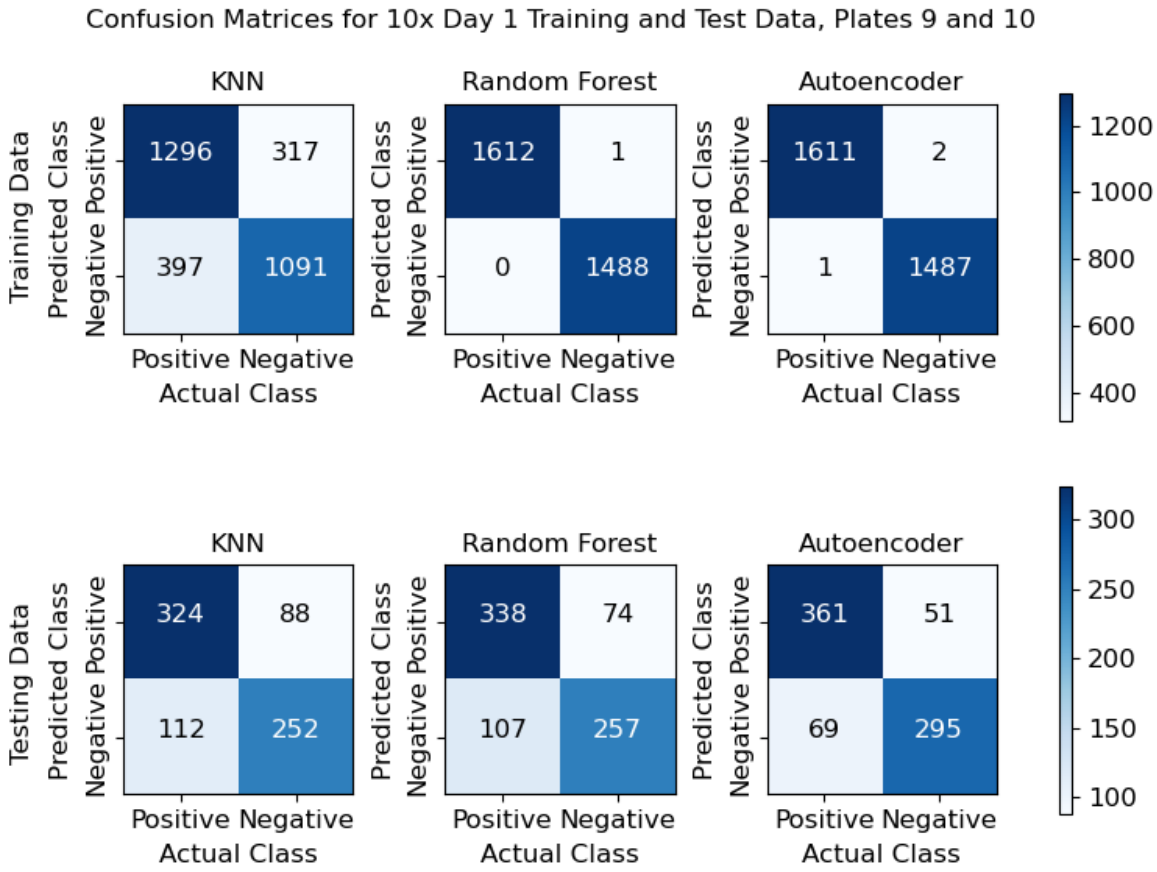
Plates 9 and 10 10x Data Figures



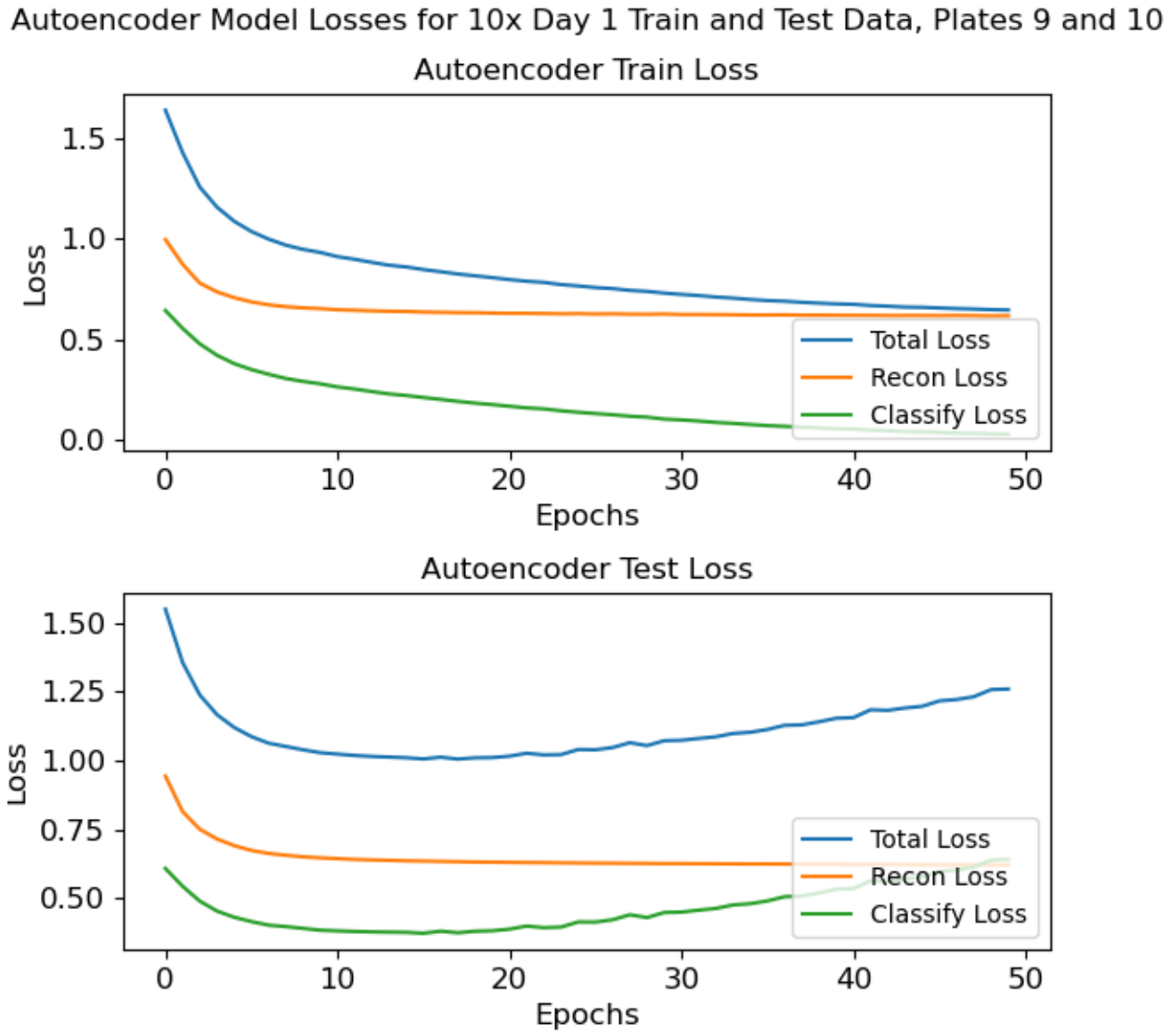
Algorithm Performances for 10x Day 1 Training and Test Data, Plates 9 and 10



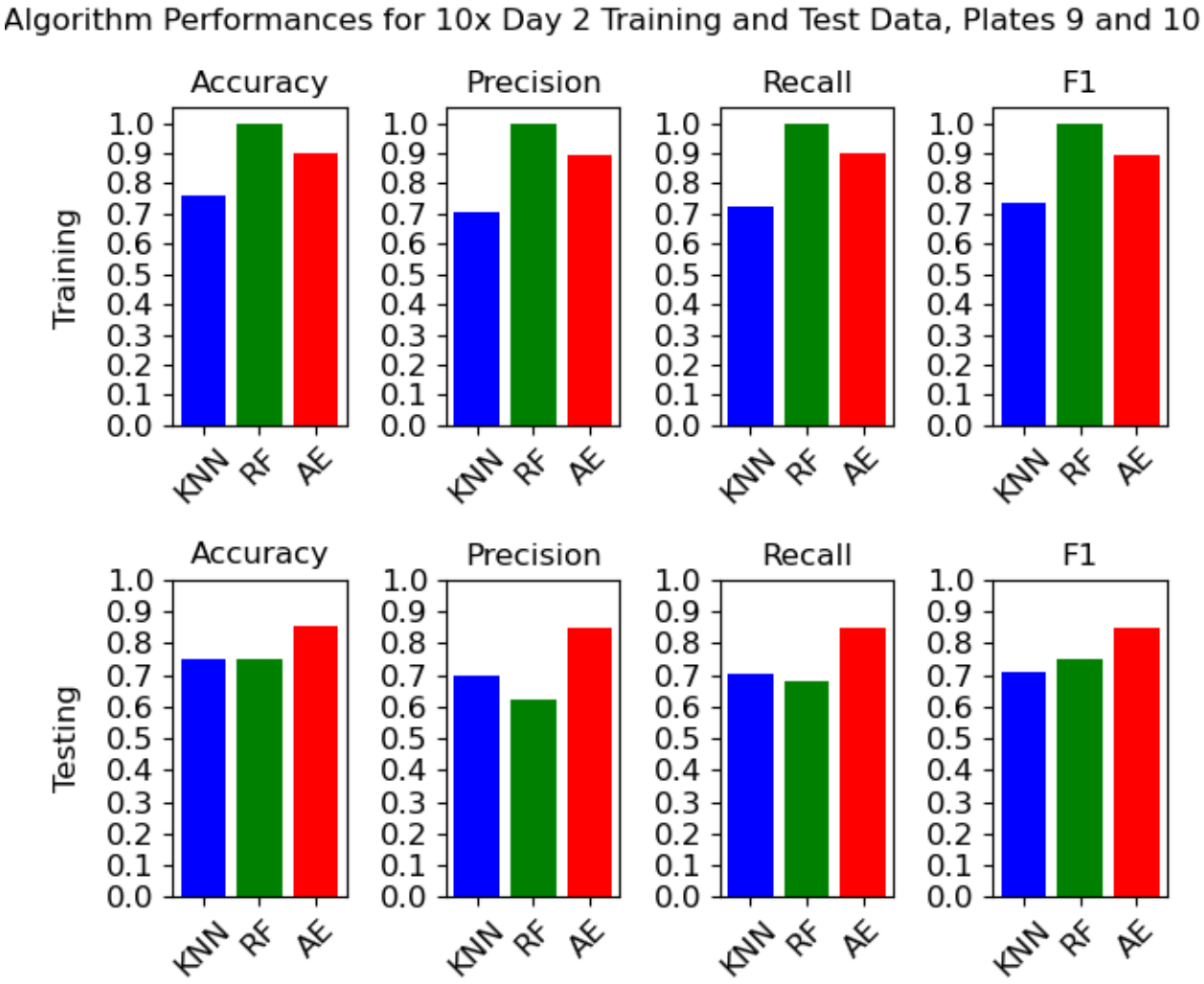
ROC Curve for 10x for Day 1 Training and Test Data, Plates 9 and 10



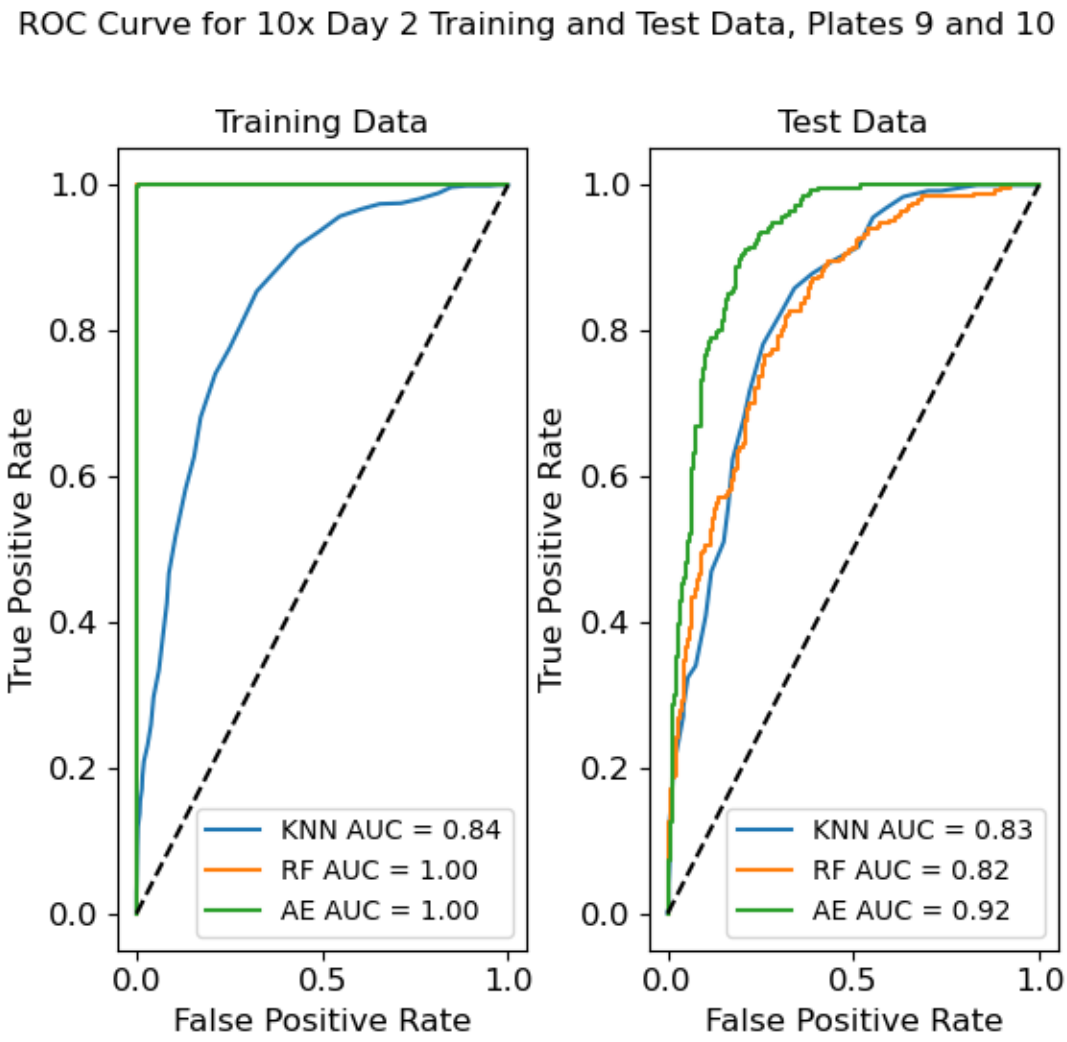
Confusion Matrices for 10x Day 1 Training and Test Data, Plates 9 and 10



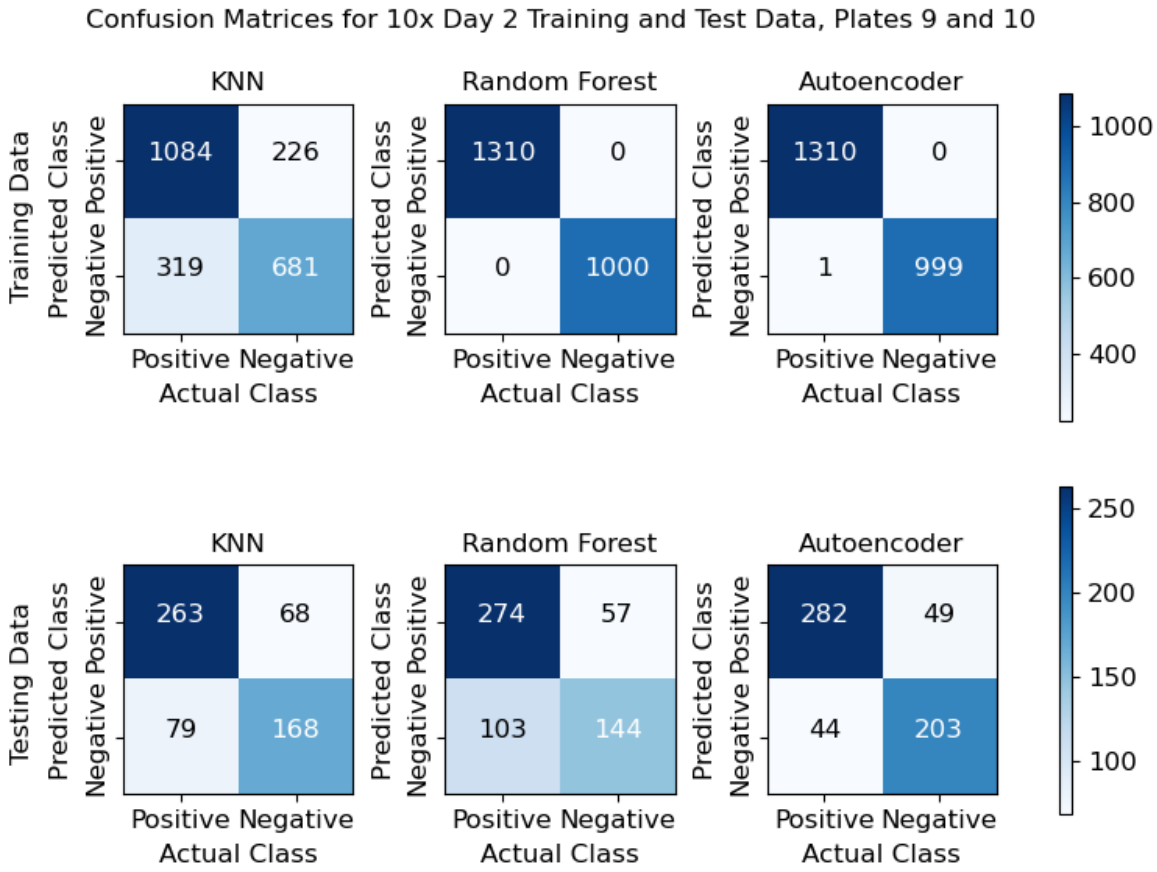
Autoencoder Model Losses for 10x Day 1 Train and Test Data, Plates 9 and 10



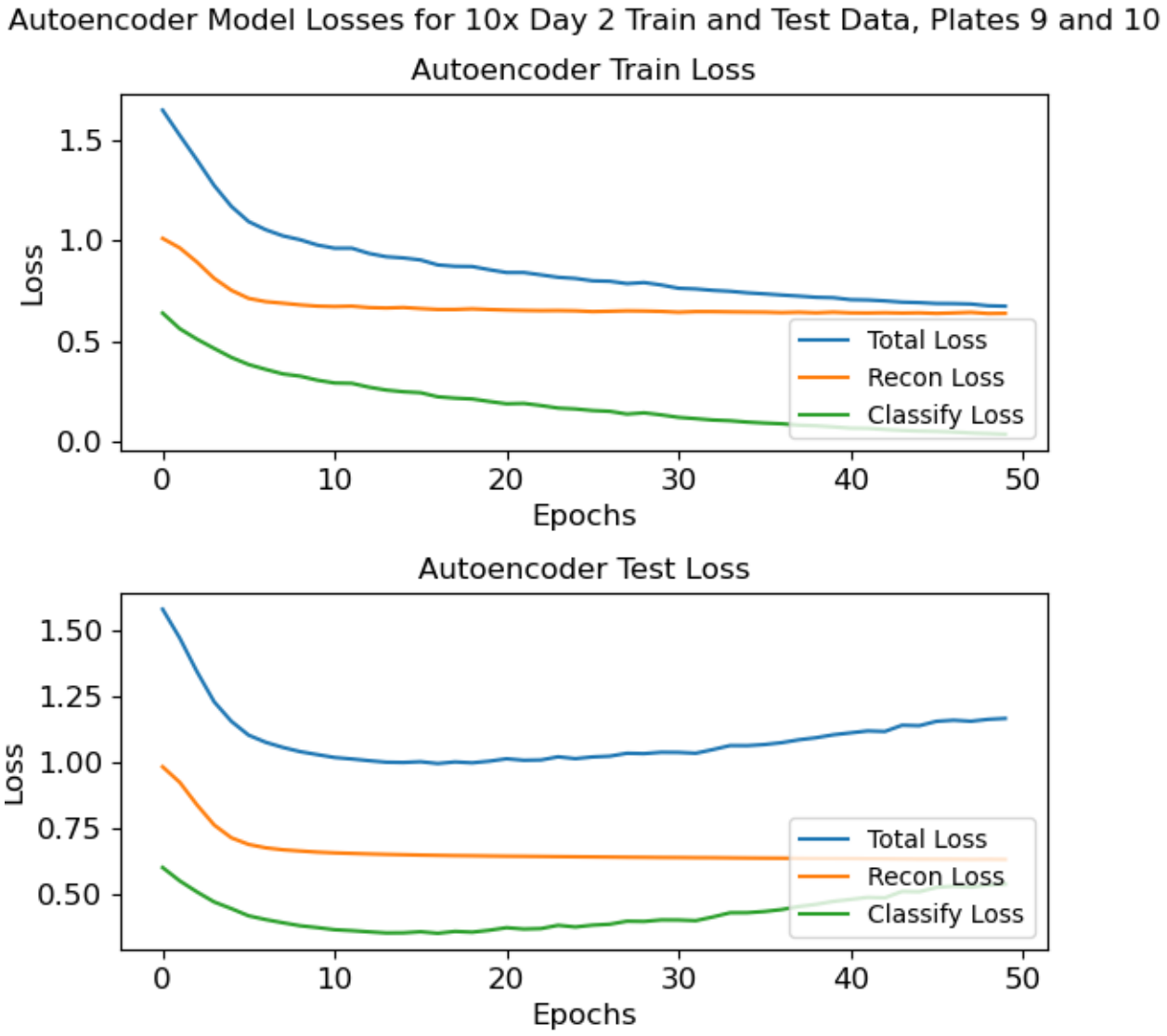
Algorithm Performances for 10x Day 2 Training and Test Data, Plates 9 and 10



ROC Curve for 10x for Day 2 Training and Test Data, Plates 9 and 10



Confusion Matrices for 10x Day 2 Training and Test Data, Plates 9 and 10



Autoencoder Model Losses for 10x Day 2 Train and Test Data, Plates 9 and 10

APPENDIX D

ADDITIONAL PYTHON CODE

Machine Learning Code

```
#%% libs
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from scipy import stats
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score

from sklearn.metrics import roc_curve, auc, confusion_matrix
from sklearn.metrics import plot_roc_curve, plot_confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, f1_score
from sklearn.metrics import ConfusionMatrixDisplay

import itertools
import torch
```

```
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from torchsummary import summary
import seaborn as sns

## for plot
import plotly.graph_objs as go
import plotly.offline as pyo
import plotly.express as px
import plotly.graph_objects as go
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation
import os
import imageio
from PIL import Image
import time
from pdb import set_trace as st

#%% utl functions

# CUDA support
print('Initializing Network...')

if torch.cuda.is_available():
    device = torch.device('cuda')
    print('run with GPU')
else:
    device = torch.device('cpu')
```

```
    print('run with CPU')


## utl functions


def get_trace(x,y,z,color='rgb(255,0,0)',name='Day1'):
    trace = go.Scatter3d(
    x=x,
    y=y,
    z=z,
    mode='markers',
    marker=dict(
        size=5,
        color=color,
        ),
    name=name,
    )
    return trace



def data_prep(df):
    '''
    input: pandas dataframe
    output: pandas dataframe
    function:
        1. remove string columns
        2. remove nan included columns
        3. z-score normalization
    '''
    df = df.dropna(axis=1)
```

```
    df.iloc[:,:-1] = z_score_normalize(df.iloc[:,:-1]) # 3
    df = df.dropna(axis=1) # drop NaNs again. z-score norm could create NaNs if column only contains 0s.

    return df

def z_score_normalize(df):
    return stats.zscore(df)



def exclude_irrelavant_columns(df):
    keys = df.keys()
    for k in keys:
        ## skip if the data type is int or float, because we need to exclude columns with strings
        if df[k].dtype == int or df[k].dtype == float:
            if df[k].isna().mean() > 0:
                df = df.drop(k,axis=1)
            else:
                continue
        else:
            df = df.drop(k,axis=1) ## axis=1 indicates drop columns

    return df

def add_label(df,label):
    w,l = df.shape
    label_array = np.ones((w,1)).astype(int) * label
    df_labeled = pd.concat([df, pd.DataFrame(label_array,columns=['label'])], axis=1)

    return df_labeled
```

```
def evaluate_metrics(model, X, y, prefix, output_folder): # for the auto encoder
    """
    Evaluate and save metrics for a given model and data.

    Parameters:
    - model: PyTorch model to evaluate
    - X: feature tensor
    - y: label tensor
    - prefix: prefix string to identify training or testing metrics
    - path: directory path to save metrics
    """
    model.eval()
    with torch.no_grad():
        outputs = model(X)
    _, predicted = torch.max(outputs, 1)
    y_true = y.numpy()
    predicted_labels = predicted.numpy()

    accuracy = accuracy_score(y_true, predicted_labels)
    precision = precision_score(y_true, predicted_labels, average='macro')
    recall = recall_score(y_true, predicted_labels, average='macro')
    f1 = f1_score(y_true, predicted_labels, average='macro')

    # print(f'{prefix} Accuracy: {accuracy}')
    # print(f'{prefix} Precision: {precision}')
    # print(f'{prefix} Recall: {recall}')
    # print(f'{prefix} F1 Score: {f1}')
```

```
# ROC, AUC, and Confusion Matrix
y_score = outputs.detach().numpy()[:, 1]  # Assuming the second column gives the score for the positive class
fpr_ae, tpr_ae, _ = roc_curve(y_true, y_score)
roc_auc_ae = auc(fpr_ae, tpr_ae)



plt.figure()
plt.plot(fpr_ae, tpr_ae, label=f'ROC curve (area = {roc_auc_ae:.2f})')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'{prefix} ROC Curve')
plt.legend()
plt.tight_layout()
plt.savefig(f'{output_folder}/{prefix}_roc.png')

confusion = confusion_matrix(y_true, predicted_labels)
# Breakdown confusion matrix into components
tn, fp, fn, tp = confusion.ravel()

# Rearrange the confusion matrix
tp, fn, fp, tn = confusion.ravel()
rearranged_confusion = np.array([[tp, fn], [fp, tn]])

# print(f'Confusion Matrix Metrics - {prefix}')
# print(f'True Positives: {tp}')
# print(f'True Negatives: {tn}')
# print(f'False Positives: {fp}')
# print(f'False Negatives: {fn}')
```

```
plt.figure()
plt.imshow(rearranged_confusion, interpolation='nearest', cmap=plt.cm.Blues)
plt.title(f'Confusion Matrix - {prefix}')
plt.colorbar()

y_classes = ['Positive', 'Negative']
x_classes = ['Positive', 'Negative']
tick_marks = np.arange(len(y_classes))
plt.xticks(tick_marks, x_classes, rotation=45)
plt.yticks(tick_marks, y_classes)
plt.ylabel('Predicted Class')
plt.xlabel('Actual Class')

fmt = 'd'
thresh = rearranged_confusion.max() / 2.
for i, j in itertools.product(range(rearranged_confusion.shape[0]), range(rearranged_confusion.shape[1])):
    plt.text(j, i, format(rearranged_confusion[i, j], fmt),
             horizontalalignment="center",
             color="white" if rearranged_confusion[i, j] > thresh else "black")
plt.tight_layout()
plt.savefig(f'{output_folder}/{prefix}_confusion.png')

 # Create output folder if it doesn't exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Create or open a text file to save metrics
metrics_file_path = os.path.join(output_folder, f"{prefix}_metrics.txt")
with open(metrics_file_path, 'w') as f:
```

```
    f.write(f"{prefix} Metrics\n")
    f.write("=" * 40 + "\n")
    f.write(f"Accuracy: {accuracy}\n")
    f.write(f"Precision: {precision}\n")
    f.write(f"Recall: {recall}\n")
    f.write(f"F1 Score: {f1}\n")
    f.write(f"ROC AUC: {roc_auc_ae}\n")
    f.write("=" * 40 + "\n")
    f.write(f"Confusion Matrix\n")
    f.write("-" * 40 + "\n")
    f.write(f"True Positives: {tp}\n")
    f.write(f"True Negatives: {tn}\n")
    f.write(f"False Positives: {fp}\n")
    f.write(f"False Negatives: {fn}\n")

    f.write("\n" + "=" * 40 + "\n")
    f.write("Autoencoder Loss Metrics (Train)\n")
    f.write("-" * 40 + "\n")
    f.write(f"Total Loss Train: {Loss_hist_train[-1,0]}\n")  # Taking the last value as the final loss
    f.write(f"Recon Loss Train: {Loss_hist_train[-1,1]}\n")
    f.write(f"Classify Loss Train: {Loss_hist_train[-1,2]}\n")

    f.write("=" * 40 + "\n")
    f.write("Autoencoder Loss Metrics (Test)\n")
    f.write("-" * 40 + "\n")
    f.write(f"Total Loss Test: {Loss_hist_test[-1,0]}\n")  # Taking the last value as the final loss
    f.write(f"Recon Loss Test: {Loss_hist_test[-1,1]}\n")
    f.write(f"Classify Loss Test: {Loss_hist_test[-1,2]}\n")
return fpr_ae, tpr_ae, roc_auc_ae, confusion
```

```
def evaluate_and_plot(clf, X_train, y_train, X_test, y_test, classifier_name, output_folder): # for KNN and RF
    # Generate the ROC curve
    y_score = clf.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_score)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=1, label=f'ROC curve (area = {roc_auc:0.2f})')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC - {classifier_name}')
    plt.legend(loc="lower right")
    plt.tight_layout()
    plt.savefig(os.path.join(output_folder, f'ROC_{classifier_name}.png'))

    # Generate the confusion matrix
    y_pred = clf.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)

    # # Print the elements of the confusion matrix
    # tn, fp, fn, tp = cm.ravel()
    # Rearrange the confusion matrix
    tp, fn, fp, tn = cm.ravel()
    rearranged_confusion = np.array([[tp, fn], [fp, tn]])

      # Save metrics to a text file
```

```
with open(os.path.join(output_folder, f"{classifier_name}_metrics.txt"), "w") as f:
    f.write(f"Classifier: {classifier_name}\n")
    f.write(f"AUC: {roc_auc:.4f}\n")
    f.write(f"True Positives: {tp}\n")
    f.write(f"True Negatives: {tn}\n")
    f.write(f"False Positives: {fp}\n")
    f.write(f"False Negatives: {fn}\n")

print(f'Confusion Matrix Metrics - {classifier_name}')
print(f'True Positives: {tp}')
print(f'True Negatives: {tn}')
print(f'False Positives: {fp}')
print(f'False Negatives: {fn}')

plt.figure()
plt.imshow(rearranged_confusion, interpolation='nearest', cmap=plt.cm.Blues)
plt.title(f'Confusion Matrix - {classifier_name}')
plt.colorbar()
y_classes = ['Positive', 'Negative']
x_classes = ['Positive', 'Negative']
tick_marks = np.arange(len(y_classes))
plt.xticks(tick_marks, x_classes, rotation=45)
plt.yticks(tick_marks, y_classes)
plt.ylabel('Predicted Class')
plt.xlabel('Actual Class')
fmt = 'd'
thresh = rearranged_confusion.max() / 2.
for i, j in itertools.product(range(rearranged_confusion.shape[0]), range(rearranged_confusion.shape[1])):
    plt.text(j, i, format(rearranged_confusion[i, j], fmt),
```

```
            horizontalalignment="center",
            color="white" if rearranged_confusion[i, j] > thresh else "black")
    plt.tight_layout()
    plt.savefig(os.path.join(output_folder, f'Confusion_Matrix_{classifier_name}.png'))
    return fpr, tpr, roc_auc

#%% data processing
## data import combine multiple file sets
# plate 5: 5/2 5/3 5/4
# plate 6: 5/17 5/18 5/19
# Plate 7, 8: 7/11 7/12 7/13
# Plate 9, 10: 8/7 8/8

## data import combine multiple file sets
GD2P_files = [
] # Add as many file paths as you have for GD2P

GD2N_files = [
] # Add as many file paths as you have for GD2N
remove_col_list = ['ImageNumber', 'ObjectNumber']

GD2P = pd.concat([pd.read_csv(file).drop(remove_col_list,axis=1) for file in GD2P_files]).reset_index(drop=True)
GD2N = pd.concat([pd.read_csv(file).drop(remove_col_list,axis=1) for file in
GD2N_files]).reset_index(drop=True)

## add label to original data
GD2P_labeled = add_label(df=GD2P, label=0)
GD2N_labeled = add_label(df=GD2N, label=1)
```

```
## concatanate two dataset into one for later operations
df = pd.concat([GD2P_labeled, GD2N_labeled], axis=0)

## preparing data for classification
df_normalized = data_prep(df)

## split X -- features and y -- labels
X = df_normalized.iloc[:,:-1]
y = df_normalized.iloc[:,-1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

output_folder =

if not os.path.exists(output_folder):
    os.makedirs(output_folder)



print(f'-----------------------------------------------')
print(f'Total dataset size: {X.shape[0]}')
print(f'GD2 POS CarT cell data: {GD2P.shape[0]}')
print(f'GD2 NEG CarT cell data: {GD2N.shape[0]}')
print(f'Training dataset size:{X_train.shape[0]}')
print(f'Test dataset size:{X_test.shape[0]}')

with open(os.path.join(output_folder, 'dataset_metrics.txt'), 'w') as f:
    f.write('-----------------------------------------------\n')
    f.write(f'Total dataset size: {X.shape[0]}\n')
    f.write(f'GD2 POS CarT cell data: {GD2P.shape[0]}\n')
```

```
    f.write(f'GD2 NEG CarT cell data: {GD2N.shape[0]}\n')
    f.write(f'Training dataset size: {X_train.shape[0]}\n')
    f.write(f'Test dataset size: {X_test.shape[0]}\n')

#%% autoencoder
# st()

X_train_arr = X_train.to_numpy()
y_train_arr = y_train.to_numpy()
X_test_arr = X_test.to_numpy()
y_test_arr = y_test.to_numpy()



# Define the size of the input data
# 157 4x 156 10x
input_size = 153
# Define the size of the bottleneck layer
latent_size = 3
# Define the size of the hidden layers in the encoder and decoder
hidden_size = 32
# Define the number of epochs to train the model for
num_epochs = 50
# Define the batch size for training the model
batch_size = 64
# Define the number of output classes
num_classes = 2

# Load the data into PyTorch tensors
X_train_tensor = torch.tensor(X_train_arr, dtype=torch.float32)
```

```
y_train_tensor = torch.tensor(y_train_arr, dtype=torch.long)

X_test_tensor = torch.tensor(X_test_arr, dtype=torch.float32)

y_test_tensor = torch.tensor(y_test_arr, dtype=torch.long)


# Create a PyTorch DataLoader for the data

data_loader_train = DataLoader(TensorDataset(X_train_tensor, y_train_tensor), batch_size=batch_size,
shuffle=True)

data_loader_test = DataLoader(TensorDataset(X_test_tensor, y_test_tensor), batch_size=len(X_test_tensor),
shuffle=False)


#

# Define the encoder and decoder networks

class Encoder(nn.Module):

    def __init__(self):

        super(Encoder, self).__init__()

        self.linear1 = nn.Linear(input_size, hidden_size)

        self.linear2 = nn.Linear(hidden_size, latent_size)


    def forward(self, x):

        x = torch.relu(self.linear1(x))

        x = self.linear2(x)

        return x


class Decoder(nn.Module):

    def __init__(self):

        super(Decoder, self).__init__()

        self.linear1 = nn.Linear(latent_size, hidden_size)

        self.linear2 = nn.Linear(hidden_size, input_size)


    def forward(self, x):
```

```
        x = torch.relu(self.linear1(x))
        x = self.linear2(x)
        return x

# Define the autoencoder as a composition of the encoder and decoder
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        self.encoder = Encoder()
        self.decoder = Decoder()

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x

# Define the combined model
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.autoencoder = Autoencoder()
        self.classifier = nn.Linear(latent_size, num_classes)

    def forward(self, x):
        x = self.autoencoder.encoder(x)
        x = self.classifier(x)
        return x

# Initialize the model
```

```
autoencoder_model = Autoencoder()
model = Model()

# summary(model, input_size=(1, 157))

# Define the loss function and optimizer
reconstruction_criterion = nn.MSELoss()
classification_criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(list(model.parameters())+list(autoencoder_model.parameters()), lr=0.0005)

Loss_hist_train = []
metric_hist_train = []
Loss_hist_test = []
metric_hist_test = []
len_train = len(data_loader_train)
# Train the model
for epoch in range(num_epochs):
    running_loss = 0.0
    running_reconstruction_loss = 0.0
    running_classification_loss = 0.0
    running_accuracy = 0.0
    running_precision = 0.0
    running_recall = 0.0
    running_f1 = 0.0
    for i, batch in enumerate(data_loader_train):
        # Get the batch of data and send it to the GPU if available
        inputs, labels_batch = batch
        inputs, labels_batch = inputs.to(device), labels_batch.to(device)
```

```
# Zero the gradients
optimizer.zero_grad()

# Forward pass

outputs_ae = autoencoder_model(inputs)
outputs_c = model(inputs)



# Calculate the reconstruction loss
reconstruction_loss = reconstruction_criterion(outputs_ae, inputs)
# Calculate the classification loss
classification_loss = classification_criterion(outputs_c, labels_batch)

# Calculate the total loss as a weighted sum of the reconstruction and classification losses
loss = reconstruction_loss + classification_loss

# Backward pass
loss.backward()

# Update the weights
optimizer.step()

# Update the running losses
running_loss += loss.item()
running_reconstruction_loss += reconstruction_loss.item()
running_classification_loss += classification_loss.item()
```

```
        predicted_labels = torch.argmax(outputs_c[:, :num_classes], dim=1).detach().numpy()


        running_accuracy += accuracy_score(labels_batch.numpy(), predicted_labels)
        running_precision += precision_score(labels_batch.numpy(), predicted_labels, average='macro',
zero_division=1)
        running_recall += recall_score(labels_batch.numpy(), predicted_labels, average='macro')
        running_f1 += f1_score(labels_batch.numpy(), predicted_labels, average='macro')


        # Print the losses every 100 batches
        # if (i+1) % 100 == 0:
    running_loss = running_loss/len_train
    running_reconstruction_loss = running_reconstruction_loss/len_train
    running_classification_loss = running_classification_loss/len_train
    running_accuracy = running_accuracy/len_train
    running_precision = running_precision/len_train
    running_recall = running_recall/len_train
    running_f1 = running_f1/len_train

    print('Epoch [{}/{}], Loss: {:.4f}, Reconstruction Loss: {:.4f}, Classification Loss: {:.4f}'.format(epoch+1,
num_epochs, running_loss, running_reconstruction_loss, running_classification_loss))
    Loss_hist_train.append((running_loss,running_reconstruction_loss,running_classification_loss))
    metric_hist_train.append((running_accuracy, running_precision, running_recall, running_f1))



    if epoch % 1 == 0:
        running_loss_test = 0.0
        running_reconstruction_loss_test = 0.0
        running_classification_loss_test = 0.0
        running_accuracy_test = 0.0
        running_precision_test = 0.0
```

```
    running_recall_test = 0.0
    running_f1_test = 0.0
    for input_test, gt_test in data_loader_test:
        input_test = input_test.to(device)
        gt_test = gt_test.to(device)

        outputs_ae_test = autoencoder_model(input_test)
        outputs_c_test = model(input_test)

        # Calculate the reconstruction loss
        reconstruction_loss_test = reconstruction_criterion(outputs_ae_test, input_test)
        # Calculate the classification loss
        classification_loss_test = classification_criterion(outputs_c_test, gt_test)

        # Calculate the total loss as a weighted sum of the reconstruction and classification losses
        loss_test = reconstruction_loss_test + classification_loss_test

        running_loss_test += loss_test.item()
        running_reconstruction_loss_test += reconstruction_loss_test.item()
        running_classification_loss_test += classification_loss_test.item()

        predicted_labels_test = torch.argmax(outputs_c_test[:, :num_classes], dim=1).detach().numpy()

        running_accuracy_test += accuracy_score(gt_test.numpy(), predicted_labels_test)
        running_precision_test += precision_score(gt_test.numpy(), predicted_labels_test, average='macro',
zero_division=1)
        running_recall_test += recall_score(gt_test.numpy(), predicted_labels_test, average='macro')
        running_f1_test += f1_score(gt_test.numpy(), predicted_labels_test, average='macro')
```

```
        print('Test:::Epoch [{}/{}], Loss: {:.4f}, Reconstruction Loss: {:.4f}, Classification Loss:
{:.4f}'.format(epoch+1, num_epochs, running_loss_test, running_reconstruction_loss_test,
running_classification_loss_test))

        Loss_hist_test.append((running_loss_test,running_reconstruction_loss_test,running_classification_loss_test))

        metric_hist_test.append((running_accuracy_test,running_precision_test,running_recall_test,running_f1_test))


Loss_hist_train = np.asarray(Loss_hist_train)

Loss_hist_test = np.asarray(Loss_hist_test)

metric_hist_train = np.asarray(metric_hist_train)

metric_hist_test = np.asarray(metric_hist_test)



plt.figure(1)

plt.title('Train Loss')

plt.plot(Loss_hist_train[:,0],label='Total Loss Train')

plt.plot(Loss_hist_train[:,1],label='Recon Loss Train')

plt.plot(Loss_hist_train[:,2],label='Classify Loss Train')

plt.legend()

plt.savefig(os.path.join(output_folder, 'autoencoder_train_loss.png'))


plt.close()

plt.figure(2)


plt.title('Test Loss')

plt.plot(Loss_hist_test[:,0],label='Total Loss Test')

plt.plot(Loss_hist_test[:,1],label='Recon Loss Test')

plt.plot(Loss_hist_test[:,2],label='Classify Loss Test')

plt.legend()

plt.savefig(os.path.join(output_folder, 'autoencoder_test_loss.png'))
```

```
plt.close()
plt.figure(3)
plt.title('Train Metrics')
plt.plot(metric_hist_train[:,0],label='Accuracy Train')
plt.plot(metric_hist_train[:,1],label='Precision Train')
plt.plot(metric_hist_train[:,2],label='Recall Train')
plt.plot(metric_hist_train[:,3],label='F1 Train')
plt.legend()
plt.savefig(os.path.join(output_folder, 'autoencoder_train_metrics.png'))
plt.close()
plt.figure(4)
plt.title('Test Metrics')
plt.plot(metric_hist_test[:,0],label='Accuracy Test')
plt.plot(metric_hist_test[:,1],label='Precision Test')
plt.plot(metric_hist_test[:,2],label='Recall Test')
plt.plot(metric_hist_test[:,3],label='F1 Test')
plt.legend()
plt.savefig(os.path.join(output_folder, 'autoencoder_test_metrics.png'))
plt.close()
# plt.show()



### training


print(f'---------- Metrics for Training ----------')


encoded_data_train = model.autoencoder.encoder(X_train_tensor).detach().numpy()
predicted_labels = torch.argmax(model(X_train_tensor)[:, :num_classes], dim=1).detach().numpy()
```

```
accuracy = accuracy_score(y_train_tensor.numpy(), predicted_labels)
precision = precision_score(y_train_tensor.numpy(), predicted_labels, average='macro')
recall = recall_score(y_train_tensor.numpy(), predicted_labels, average='macro')
f1 = f1_score(y_train_tensor.numpy(), predicted_labels, average='macro')

print('Training Accuracy: {:.4f}%'.format(accuracy*100))
print('Training Precision: {:.4f}%'.format(precision*100))
print('Training Recall: {:.4f}%'.format(recall*100))
print('Training F1: {:.4f}%'.format(f1*100))

label1_ind = np.where(y_train_tensor.numpy()==0)
label2_ind = np.where(y_train_tensor.numpy()==1)



time.sleep(2)

# st()

# ### testing

print(f'---------- Metrics for Testing ----------')

encoded_data_test = model.autoencoder.encoder(X_test_tensor).detach().numpy()
predicted_labels = torch.argmax(model(X_test_tensor)[:, :num_classes], dim=1).detach().numpy()
accuracy = accuracy_score(y_test_tensor.numpy(), predicted_labels)
precision = precision_score(y_test_tensor.numpy(), predicted_labels, average='macro')
recall = recall_score(y_test_tensor.numpy(), predicted_labels, average='macro')
f1 = f1_score(y_test_tensor.numpy(), predicted_labels, average='macro')
```

```
print('Test Accuracy: {:.4f}%'.format(accuracy*100))

print('Test Precision: {:.4f}%'.format(precision*100))

print('Test Recall: {:.4f}%'.format(recall*100))

print('Test F1: {:.4f}%'.format(f1*100))


label1_ind = np.where(y_test_tensor.numpy()==0)

label2_ind = np.where(y_test_tensor.numpy()==1)


# Create the figure and 3D axes



# Create the figure and 3D axes static plot


fig = plt.figure(figsize=(6.5, 4))


plt.suptitle('Autoencoder Classification for All 10x Day 1 Train and Test Data', fontsize=12, y=0.93)


ax1 = fig.add_subplot(121, projection='3d')


# Assuming 'encoded_data' is your 3D data

ax1.scatter(encoded_data_train[label1_ind][:,0], encoded_data_train[label1_ind][:,1],
encoded_data_train[label1_ind][:,2], c='r', label='GD2 POS')

ax1.scatter(encoded_data_train[label2_ind][:,0], encoded_data_train[label2_ind][:,1],
encoded_data_train[label2_ind][:,2], c='b', label='GD2 NEG')

ax1.set_xlabel('X')  # Replace 'X Label' with your actual label

ax1.set_ylabel('Y')  # Replace 'Y Label' with your actual label

ax1.set_zlabel('Z')  # Replace 'Z Label' with your actual label

ax1.set_title('Training Data', fontsize=12)

ax1.view_init(elev=15, azim=15)

ax1.legend(fontsize=12)
```

```
# plt.savefig(os.path.join(output_folder, 'autoencoder_train_scatter.png'))


#test subplot

ax2 = fig.add_subplot(122, projection='3d')


# Assuming 'encoded_data' is your 3D data

ax2.scatter(encoded_data_test[label1_ind][:,0], encoded_data_test[label1_ind][:,1],
encoded_data_test[label1_ind][:,2], c='r', label='GD2 POS')

ax2.scatter(encoded_data_test[label2_ind][:,0], encoded_data_test[label2_ind][:,1],
encoded_data_test[label2_ind][:,2], c='b', label='GD2 NEG')

ax2.set_xlabel('X')  # Replace 'X Label' with your actual label

ax2.set_ylabel('Y')  # Replace 'Y Label' with your actual label

ax2.set_zlabel('Z')  # Replace 'Z Label' with your actual label

ax2.set_title('Test Data', fontsize=12)

ax2.view_init(elev=15, azim=15)

ax2.legend(fontsize=12)


plt.savefig(os.path.join(output_folder, 'autoencoder_scatter_plot.png'))

plt.close()


# Update function for each frame of the animation

def update(frame):

    ax1.view_init(elev=15, azim=frame)  # Change the view angle

    ax2.view_init(elev=15, azim=frame)

# Create the animation

animation = FuncAnimation(fig, update, frames=np.linspace(0, 360, 360), interval=100)


# Save the animation as a GIF file
```

```
animation_file_path = os.path.join(output_folder, 'autoencoder_test_train_data.gif')

animation.save(animation_file_path, writer='pillow')

# animation.save('G:/CellProfilerPipelines/CART_PROJECT/tempfiledump//autoencoder_train_data.gif',
writer='pillow')

plt.close()


# Assume y_test_tensor.numpy() is your ground-truth label and predicted_labels is your predicted label

# y_test_prob is the predicted probabilities

evaluate_metrics(model, X_train_tensor, y_train_tensor, 'Autoencoder_Training', output_folder)

plt.close()

plt.close()


evaluate_metrics(model, X_test_tensor, y_test_tensor, 'Autoencoder_Testing', output_folder)

plt.close()

plt.close()

#%% PCA

# perform PCA

pca = PCA(n_components=int(X.shape[1]/2)) ## get half of features as PCs

pca.fit(X)


# transform the data

transformed_X = pca.transform(X)


# ## plot for PCA

GD2P_index = np.where(y==0)[0]

GD2N_index = np.where(y==1)[0]

# plt.figure('PCA')

# plt.plot(transformed_X[day_1_index,1],transformed_X[day_1_index,2],'bo',label='DAY-1')

# plt.plot(transformed_X[day_5_index,1],transformed_X[day_5_index,2],'rx',label='DAY-5')
```

```
# plt.show()


X_train, X_test, y_train, y_test = train_test_split(transformed_X, y, test_size=0.20, random_state=42)


#%% KNN


# Initialize metrics lists
K_list = np.arange(1, 50)
acc_train_list_knn, acc_test_list_knn = [], []
recall_train_list_knn, recall_test_list_knn = [], []
f1_train_list_knn, f1_test_list_knn = [], []
precision_train_list_knn, precision_test_list_knn = [], []


# Loop over different values of k
for k in K_list:
    neigh = KNeighborsClassifier(n_neighbors=k)
    neigh.fit(X_train, y_train)
    train_predict = neigh.predict(X_train)
    test_predict = neigh.predict(X_test)


    # Compute and store metrics
    acc_train_list_knn.append(accuracy_score(y_train, train_predict))
    acc_test_list_knn.append(accuracy_score(y_test, test_predict))
    recall_train_list_knn.append(recall_score(y_train, train_predict))
    recall_test_list_knn.append(recall_score(y_test, test_predict))
    f1_train_list_knn.append(f1_score(y_train, train_predict))
    f1_test_list_knn.append(f1_score(y_test, test_predict))
    precision_train_list_knn.append(precision_score(y_train, train_predict))
    precision_test_list_knn.append(precision_score(y_test, test_predict))
```

```
# Create plots
plt.figure('KNN Performances', figsize=(12, 8))

metrics_list = ['Accuracy', 'Recall', 'F1 Score', 'Precision']
knn_train_metrics = [acc_train_list_knn, recall_train_list_knn, f1_train_list_knn, precision_train_list_knn]
knn_test_metrics = [acc_test_list_knn, recall_test_list_knn, f1_test_list_knn, precision_test_list_knn]

for i, metric in enumerate(metrics_list):
    plt.subplot(2, 2, i + 1)
    plt.title(f'{metric} Performance')
    plt.xlabel('k')
    plt.ylabel(metric)
    plt.plot(knn_train_metrics[i], label='Train')
    plt.plot(knn_test_metrics[i], label='Test')
    plt.legend()

plt.suptitle('KNN Performance')
plt.tight_layout()
plt.savefig(os.path.join(output_folder, 'KNN_Performances_Multiple_Metrics.png'))
plt.close()
# Initialize lists to store KNN error rates
knn_train_error = []
knn_test_error = []

# KNN Error Calculation
for k in K_list:
    neigh = KNeighborsClassifier(n_neighbors=k)
    neigh.fit(X_train, y_train)
```

```
    train_predict = neigh.predict(X_train)
    test_predict = neigh.predict(X_test)

    knn_train_error.append(1 - accuracy_score(y_train, train_predict))
    knn_test_error.append(1 - accuracy_score(y_test, test_predict))

# Find best k
k_best = np.argmax(np.array(f1_test_list_knn))
# Train and evaluate model with best k
neigh_best = KNeighborsClassifier(n_neighbors=k_best)
neigh_best.fit(X_train, y_train)
train_predict = neigh_best.predict(X_train)
test_predict = neigh_best.predict(X_test)
evaluate_and_plot(neigh_best, X_train, y_train, X_test, y_test, 'KNN_Testing', output_folder)
plt.close()
plt.close()
evaluate_and_plot(neigh_best, X_train, y_train, X_train, y_train, 'KNN_Training', output_folder)
plt.close()
plt.close()
# Create a 1x2 subplot grid
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# GD2 Positive and GD2 Negative indices for training data predictions
GD2P_train_index = np.where(train_predict == 0)[0]
GD2N_train_index = np.where(train_predict == 1)[0]

# GD2 Positive and GD2 Negative indices for testing data predictions
GD2P_test_index = np.where(test_predict == 0)[0]
GD2N_test_index = np.where(test_predict == 1)[0]
```

```
# Scatter plot for training data predictions

axes[0].scatter(X_train[GD2P_train_index, 1], X_train[GD2P_train_index, 2], c='b', alpha=0.5, label='Predicted GD2P Train')

axes[0].scatter(X_train[GD2N_train_index, 1], X_train[GD2N_train_index, 2], c='r', alpha=0.5, label='Predicted GD2N Train')

axes[0].set_title('KNN Classification for Training Data')

axes[0].legend()


# Scatter plot for testing data predictions

axes[1].scatter(X_test[GD2P_test_index, 1], X_test[GD2P_test_index, 2], c='b', alpha=0.5, label='Predicted GD2P Test')

axes[1].scatter(X_test[GD2N_test_index, 1], X_test[GD2N_test_index, 2], c='r', alpha=0.5, label='Predicted GD2N Test')

axes[1].set_title('KNN Classification for Testing Data')

axes[1].legend()


# Save the figure

plt.savefig(os.path.join(output_folder, "two_panel_KNN_classification_plot.png"))

plt.close()

# Display performance

print(f'KNN performance')

print(f'Best k is {k_best} corresponding metrics:')

print(f'Acc: train: {accuracy_score(y_train, train_predict)}, test: {accuracy_score(y_test, test_predict)}')

print(f'Recall: train: {recall_score(y_train, train_predict)}, test: {recall_score(y_test, test_predict)}')

print(f'F1: train: {f1_score(y_train, train_predict)}, test: {f1_score(y_test, test_predict)}')

print(f'Precision: train: {precision_score(y_train, train_predict)}, test: {precision_score(y_test, test_predict)}')


with open(os.path.join(output_folder, 'KNN_Training_metrics.txt'), 'a') as f:

    f.write("KNN Performance\n")

    f.write("-" * 40 + "\n")
```

```
    f.write(f"Best k is {k_best} corresponding metrics:\n")
    f.write(f"Acc: train: {accuracy_score(y_train, train_predict):.4f}\n")
    f.write(f"Recall: train: {recall_score(y_train, train_predict):.4f}\n")
    f.write(f"F1: train: {f1_score(y_train, train_predict):.4f}\n")
    f.write(f"Precision: train: {precision_score(y_train, train_predict):.4f}\n")
    f.write("-" * 40 + "\n")
    f.write("KNN Error Metrics\n")
    f.write(f"Training Error (last value): {knn_train_error[-1]:.4f}\n")

with open(os.path.join(output_folder, 'KNN_Testing_metrics.txt'), 'a') as f:
    f.write("KNN Performance\n")
    f.write("-" * 40 + "\n")
    f.write(f"Best k is {k_best} corresponding metrics:\n")
    f.write(f"Acc: test: {accuracy_score(y_test, test_predict):.4f}\n")
    f.write(f"Recall: test: {recall_score(y_test, test_predict):.4f}\n")
    f.write(f"F1: test: {f1_score(y_test, test_predict):.4f}\n")
    f.write(f"Precision: test: {precision_score(y_test, test_predict):.4f}\n")
    f.write("-" * 40 + "\n")
    f.write("KNN Error Metrics\n")
    f.write(f"Testing Error (last value): {knn_test_error[-1]:.4f}\n")

#%% Random Forest

max_depth_list = np.arange(1, 50)
acc_train_list_rf, acc_test_list_rf = [], []
recall_train_list_rf, recall_test_list_rf = [], []
f1_train_list_rf, f1_test_list_rf = [], []
precision_train_list_rf, precision_test_list_rf = [], []
```

```
# Iterate through different max depths
for md in max_depth_list:
    clf = RandomForestClassifier(max_depth=md, random_state=0)
    clf.fit(X_train, y_train)
    train_predict = clf.predict(X_train)
    test_predict = clf.predict(X_test)

    # Calculate metrics for both training and test data
    acc_train_list_rf.append(accuracy_score(y_train, train_predict))
    acc_test_list_rf.append(accuracy_score(y_test, test_predict))
    recall_train_list_rf.append(recall_score(y_train, train_predict))
    recall_test_list_rf.append(recall_score(y_test, test_predict))
    f1_train_list_rf.append(f1_score(y_train, train_predict))
    f1_test_list_rf.append(f1_score(y_test, test_predict))
    precision_train_list_rf.append(precision_score(y_train, train_predict))
    precision_test_list_rf.append(precision_score(y_test, test_predict))

# Plot all the metrics
plt.figure('Random Forest Performances', figsize=(12, 8))

# Initialize lists to store Random Forest error rates
rf_train_error = []
rf_test_error = []

# Random Forest Error Calculation
for md in max_depth_list:
    clf = RandomForestClassifier(max_depth=md, random_state=0)
    clf.fit(X_train, y_train)
    train_predict = clf.predict(X_train)
```

```
    test_predict = clf.predict(X_test)

    rf_train_error.append(1 - accuracy_score(y_train, train_predict))
    rf_test_error.append(1 - accuracy_score(y_test, test_predict))

rf_train_metrics = [acc_train_list_rf, recall_train_list_rf, f1_train_list_rf, precision_train_list_rf]

rf_test_metrics = [acc_test_list_rf, recall_test_list_rf, f1_test_list_rf, precision_test_list_rf]

metric_names = ['Accuracy', 'Recall', 'F1-Score', 'Precision']

for i, metric_name in enumerate(metric_names):
    plt.subplot(2, 2, i+1)
    plt.title(f'{metric_name} Performance')
    plt.xlabel('max depth')
    plt.ylabel(metric_name)
    plt.plot(rf_train_metrics[i], label='Train')
    plt.plot(rf_test_metrics[i], label='Test')
    plt.legend()

plt.suptitle('Random Forest Performance')
plt.tight_layout()
# Save the plot
plt.savefig(os.path.join(output_folder, 'Random_Forest_Performances.png'))
plt.close()
# Find best max depth
md_best = np.argmax(np.array(f1_test_list_rf))

# Train best model
```

```
clf_best = RandomForestClassifier(max_depth=md_best, random_state=0)

clf_best.fit(X_train, y_train)

train_predict = clf_best.predict(X_train)

test_predict = clf_best.predict(X_test)



# Create a 1x2 subplot grid

fig, axes = plt.subplots(1, 2, figsize=(14, 6))


# GD2 Positive and GD2 Negative indices for training data predictions

GD2P_train_index = np.where(train_predict == 0)[0]

GD2N_train_index = np.where(train_predict == 1)[0]


# GD2 Positive and GD2 Negative indices for testing data predictions

GD2P_test_index = np.where(test_predict == 0)[0]

GD2N_test_index = np.where(test_predict == 1)[0]


# Scatter plot for training data predictions

axes[0].scatter(X_train[GD2P_train_index, 1], X_train[GD2P_train_index, 2], c='b', alpha=0.5, label='Predicted
GD2P Train')

axes[0].scatter(X_train[GD2N_train_index, 1], X_train[GD2N_train_index, 2], c='r', alpha=0.5, label='Predicted
GD2N Train')

axes[0].set_title('Random Forest Classification for Training Data')

axes[0].legend()


# Scatter plot for testing data predictions

axes[1].scatter(X_test[GD2P_test_index, 1], X_test[GD2P_test_index, 2], c='b', alpha=0.5, label='Predicted GD2P
Test')

axes[1].scatter(X_test[GD2N_test_index, 1], X_test[GD2N_test_index, 2], c='r', alpha=0.5, label='Predicted GD2N
Test')

axes[1].set_title('Random Forest Classification for Testing Data')
```

```
axes[1].legend()

# Save the figure
plt.savefig(os.path.join(output_folder, "two_panel_rf_classification_plot.png"))
plt.close()
# Your evaluate_and_plot function calls
evaluate_and_plot(clf_best, X_train, y_train, X_test, y_test, 'Random_Forest_Testing', output_folder)
plt.close()
plt.close()
evaluate_and_plot(clf_best, X_train, y_train, X_train, y_train, 'Random_Forest_Training', output_folder)
plt.close()
plt.close()

# Show performance metrics
print(f'Random Forest performance')
print(f'Best max depth is {md_best} corresponding metrics:')
print(f'Acc: train: {accuracy_score(y_train, train_predict)}, test: {accuracy_score(y_test, test_predict)}')
print(f'Recall: train: {recall_score(y_train, train_predict)}, test: {recall_score(y_test, test_predict)}')
print(f'F1: train: {f1_score(y_train, train_predict)}, test: {f1_score(y_test, test_predict)}')
print(f'Precision: train: {precision_score(y_train, train_predict)}, test: {precision_score(y_test, test_predict)}')

with open(os.path.join(output_folder, 'Random_Forest_Training_metrics.txt'), 'a') as f:
    f.write("Random Forest performance\n")
    f.write(f"Best max depth is {md_best} corresponding metrics:\n")
    f.write(f"Acc: train: {accuracy_score(y_train, train_predict):.4f}\n")
    f.write(f"Recall: train: {recall_score(y_train, train_predict):.4f}\n")
    f.write(f"F1: train: {f1_score(y_train, train_predict):.4f}\n")
    f.write(f"Precision: train: {precision_score(y_train, train_predict):.4f}\n")
    f.write("-" * 40 + "\n")
```

```
    f.write("Random Forest Error Metrics\n")
    f.write("-" * 40 + "\n")
    f.write(f"Training Error (last value): {rf_train_error[-1]:.4f}\n")

with open(os.path.join(output_folder, 'Random_Forest_Testing_metrics.txt'), 'a') as f:
    f.write("Random Forest performance\n")
    f.write(f"Best max depth is {md_best} corresponding metrics:\n")
    f.write(f"Acc: test: {accuracy_score(y_test, test_predict):.4f}\n")
    f.write(f"Recall: test: {recall_score(y_test, test_predict):.4f}\n")
    f.write(f"F1: train: test: {f1_score(y_test, test_predict):.4f}\n")
    f.write(f"Precision: test: {precision_score(y_test, test_predict):.4f}\n")
    f.write("-" * 40 + "\n")
    f.write("Random Forest Error Metrics\n")
    f.write("-" * 40 + "\n")
    f.write(f"Testing Error (last value): {rf_test_error[-1]:.4f}\n")
```

DPC Reconstruction Code

```
# -*- coding: utf-8 -*-
"""
Created on Tue Dec 22 17:15:11 2020

@author: nas44244
"""
#%%
from PIL import Image
from dpc_algorithm import DPCSolver
import os
import numpy as np
from matplotlib import pyplot as plt
from time import perf_counter

#%% load files

imDir='.\\savedFrames\\DPC'
outDir=imDir+os.path.sep+'phaseRecon'
fList=os.listdir(imDir)

fList=[fn for fn in fList if fn.endswith('.tif')]
if not os.path.exists(outDir): os.mkdir(outDir)

#%% run parameters

# DPC parameters
wavelength     =  0.530 #micron
```

```
mag            =   2.03*4
na             =   0.13 #numerical aperture
na_in          =    0.0
pixel_size_cam =    6.5 #pixel size of camera
dpc_num        =      4 #number of DPC images captured for each absorption and phase frame
pixel_size     = pixel_size_cam/mag
rotation       = np.array([270,90,180,0])+5


#%%
plt.close('all')
createSolver = True
t1=[]
print('Processing {} images for DPC reconstruction...'.format(len(fList)))
for i in range(0,1):#len(fList)):
    im=Image.open(imDir+os.path.sep+fList[i])
    print(i,'.',fList[i],'\t',im.n_frames,end='\t:\t')
    if im.n_frames !=4:
        print('')
        continue
    dpc_images = []
    t0=[]
    t0.append(perf_counter())
    for j in range(im.n_frames):
        im.seek(j)
        dpc_images.append(np.array(im.getdata()).reshape(im.size))
    t0.append(perf_counter())
    if createSolver:
        dpc_images=np.array(dpc_images)
```

```
        dpc_solver_obj = DPCSolver(dpc_images, wavelength, na, na_in, pixel_size, rotation,
dpc_num=dpc_num,imgInit=False)

        dpc_solver_obj.setTikhonovRegularization(reg_u = 1e-1, reg_p = 5e-3)

        dpc_solver_obj.preSolvePhase()

        createSolver = False

    for j in range(len(dpc_images)): dpc_solver_obj.updateImg(dpc_images[j],j)

    t0.append(perf_counter())

    phase=dpc_solver_obj.solvePhase()

    t0.append(perf_counter())

    print('\t\t'.join(['{:.3f}'.format((t0[ti+1]-t0[ti])) for ti in range(len(t0)-1)]))

#    plt.figure(fList[i])

#    plt.imshow(phase,cmap='gray')

    imr=Image.fromarray(phase.astype('float32'))

#    imr.save(outDir+os.path.sep+fList[i])


#    plt.title()

#    for fi in range(4):

#        t3=perf_counter()
```