Image Classification: Handwritten Japanese Hiragana Characters

Final Project

Samuel Jimenez Canizal
Tandon School of Engineering
New York University
Woodside, New York
sj3906@nyu.edu

I. INTRODUCTION

This project uses the Handwritten Japanese Hiragana Characters dataset, obtained from Kaggle, which contains 100 500x400 images of handwritten examples for each basic character in Hiragana, making it a total of 4600 examples[1]. These examples were split up and used for training, validation, and testing. While the dataset itself did not come with pre-existing features, 11 features were engineered and extracted from each image. These features include: mean pixel intensity, intensity standard deviation, minimum pixel intensity, the number of ink pixels, the bounding box's height, width, and area, the aspect ratio, the coordinates of the center of mass, and laplacian variance. The target variable was the correct hiragana character. In this case, there are 46 classes in which the models could have predicted from, which include characters like: 'あ' (aa), 'い' (ii), 'う' (uu), 'え' (ee), 'お' (oo), etc.

Thus, being a multi-class classification problem, the task of each model was to correctly classify a handwritten character sample into one of the 46 classes. This type of classification problem is interesting because visually, Hiragana characters share similar structures in the overall curvature and stroke pattern of some characters, and since there are 46 different classes that the models much predict an image belongs to, the features extracted, models used, and hyperparameters chosen must be meaningful. Additionally, some characters are written differently depending on the person. This was particularly noticeable with 'さ' (sa) where some images connected the top and bottom portions of the characters, where others left them separated.

In this project, the following models were baselined in order to get the preliminary metrics and overall accuracy, and experimented on by tuning hyperparameters and by transforming features: Logistic Regression, K-Nearest Neighbors, and Neural Networks.

II. EXPLORATORY ANALYSIS

To begin, since the dataset did not come with predetermined features, feature extraction on every image example for every class needed to be performed. However, the images were first resized from 500x400 to 28x28, normalized, and grayscale in order to make feature extraction more effective, as seen in Figure 1.

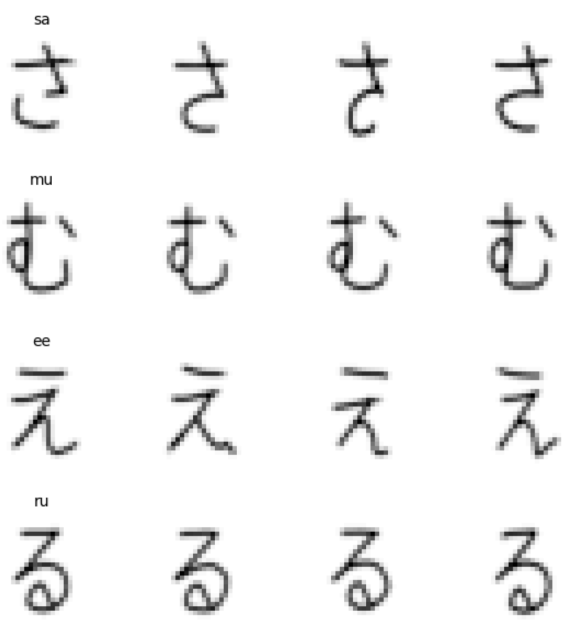

Fig.1. Examples of 28x28 Handwritten Hiragana Characters

Initially, 22 features were extracted and a correlation matrix was created from said features. However, of the 22 original features, many features were found to be redundant and the same across every example, so they were dropped and the features selected were eventually dropped to 11, which are the ones previously mentioned.

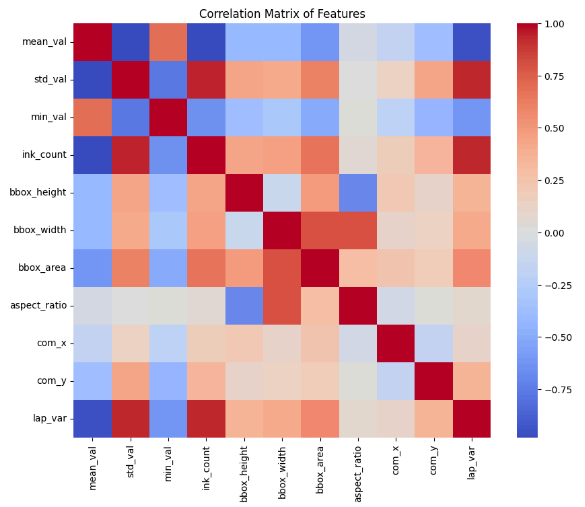

Fig. 2. Correlation Matrix of 11 Extracted Features

As seen in Figure 2, there were still some features that were either negatively or positively correlated with each other, but the element of redundancy that was previously present was, no longer was. Every correlation was to be expected and reflected the meaningful structure associated with said features.

In order to understand the differences and similarities between the features of each character/class, density plots were created. As seen in Figure 3, the density plots revealed that there is variation in handwriting and the darkness of some strokes in comparison to others. Additionally, particularly regarding the bounding box dimensions, it suggests variation in geometric shapes. While some characters may be taller than others, others were wider and more circular. The center of mass coordinates suggest that the center for each character, regarding ink intensity, was relatively the same. Finally, the laplacian variance showed some variance, reflecting the differences between characters with more curvature and intersecting strokes. Thus, from the density plots, variance between features across the examples of each class showed that no feature across each example was constant, but rather each proved to be meaningful.

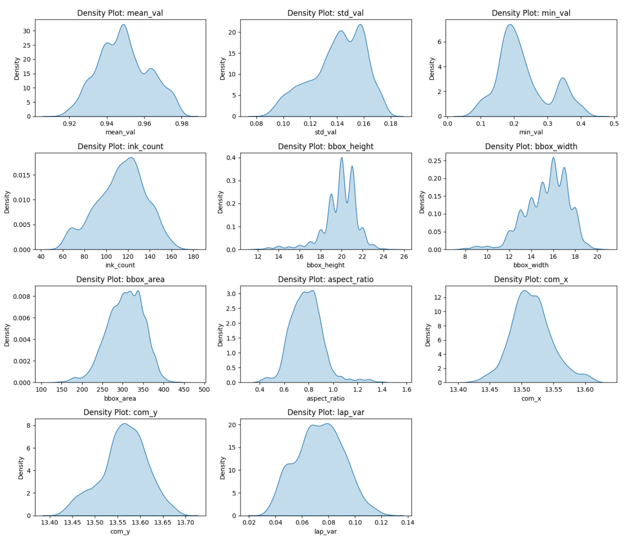

Fig. 3. Feature Density Plots

In order to confirm the findings, mean feature value bar graphs were created to visualize the difference in said feature values for each individual class. As seen in Figure 4, the variation presented in the density plots reflected the difference in mean values for each feature of each class where some values were relatively similar and other values displayed differences across classes.

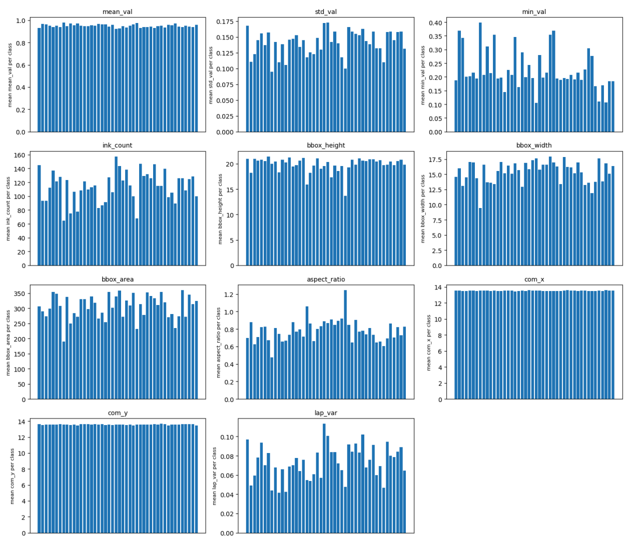

Fig. 4. Mean Feature Values Across Each Class

After visualizing feature distributions and noticing the pattern and structures between features, the data was ready to be preprocessed. First, the data was split into a training set to fit the model, a validation set to make necessary and experimental adjustments to hyperparameter values, and a testing set which was run on the model with the best metrics as a way to test the best model's ability to generalize to unseen data. 70% of the data was used for training, 15% was used for validation, and 15% was used for the test set. For the untransformed feature approach, the features were scaled using the StandardScaler() from sklearn.preprocessing was used, whereas for feature transformations, PolynomialFeatures() from sklearn.preprocessing and KMeans() from sklearn.cluster was used. Each feature varies based on values and overall scale, so StandardScaler() standardizes each feature by subtracting the mean value and scaling it to the feature variance, ensuring each feature is using a comparable scale. PolynomialFeatures() attempts to capture the nonlinearity of the multi-class classification problem and creates additional nonlinear features, whether it is of degree two or higher. KMeans() groups samples into clusters and assigns each sample a new cluster by minimizing the distance to the center. After scaling and transforming each feature using the previously mentioned methods, the following table of dimensions was created.

TABLE I
SCALING AND FEATURE TRANSFORMATION
DATA SHAPE

Feature Representation	**Train Shape**	**Validation Shape**	**Test Shape**
Scaled (Untransformed)	(3220, 11)	(690, 11)	(690, 11)
Polynomial Degree 2	(3220, 77)	(690, 77)	(690, 77)
Polynomial Degree 3	(3220, 363)	(690, 363)	(690, 363)
K-Means Clusters	(3220, 19)	(690, 19)	(690, 19)

As seen from Table 1, the data follow a pattern of 3220 examples for the training set and 690 for the validation and testing set each. However, the differences arise from the number of features due to the scaling and feature transformations. For standard untransformed scaling, the data maintains the original 11 features, whereas for a feature transformation of polynomial degree two and three, the features increase to 77 and 363 respectively. For K-Means clustering, the number of features increases to 19 to account for the original 11 features and the k value used throughout each model, k = 8. The target vector dimensions did not change and have dimensions that reflect the number of examples per set.

$$y_{train} \in \mathbb{R}^{3220}$$

$$y_{val} \in \mathbb{R}^{690}$$

$$y_{test} \in \mathbb{R}^{690}$$

After the preprocessing process was done, it was time to start training the models.

III. SUPERVISED MODELING

For each of the following models, a basic sequence of events were followed. Firstly, a baseline model was created and the validation set was used to show the preliminary metrics of the model, which include accuracy, precision, recall, and F1 score. Additionally, a confusion matrix was produced. Then, the model is continuously tested on with different hyperparameters. The metrics for each hyperparameter is saved in order to create a line graph comparing each metric with the hyperparameter setting and a table showing the metrics. Afterwards, this process was repeated for transformed features. At the end, the best model amongst all model approaches was chosen to use the test set on, seeing how the best model after experimentation generalized on unseen data.

For each model, the target was encoded between a value of 0 and 45, reflecting the number of basic Hiragana characters.

A. LOGISTIC REGRESSION

For logistic regression, LogisticRegression() from sklearn.linear_model was used. Logistic Regression was chosen because it is a simple model that can learn multiple linear decision boundaries, separating the data into multiple classes that is needed for this multiclass classification problem. The baseline logistic regression model used had no regularization, in order to get the baseline metrics and a direction to how said metrics could improve. A confusion matrix was produced and showed the following:

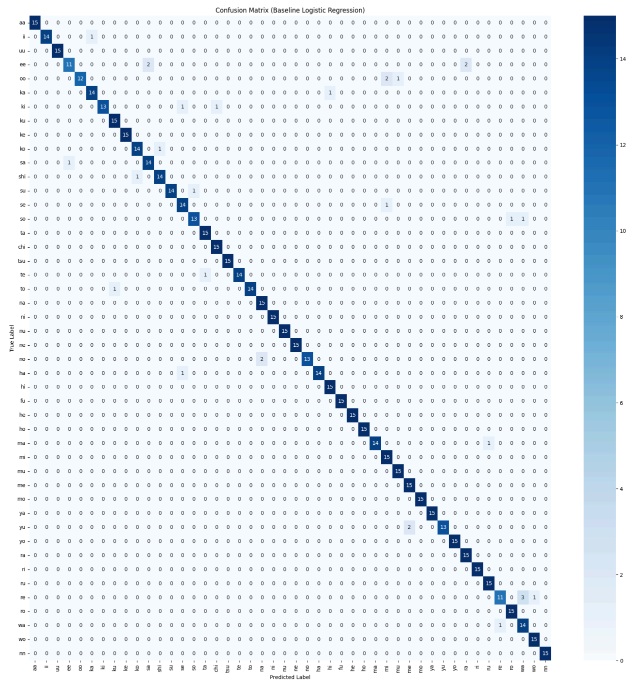

Fig. 5. Confusion Matrix for Baseline Logistic Regression Model

Figure 5 showed a high number of examples that were categorized accurately, which is reflected in Table 2.

TABLE II
METRICS FOR BASELINE LOGISTIC REGRESSION

Metric	Score
Training Accuracy	0.9792
Validation Accuracy	0.9551
Validation Precision	0.9580
Validation Recall	0.9551
Validation F1 Score	0.9546

The baseline model achieves high performance and suggests good generalization, however the results also suggest room for improvement in possible non-linear relationships/structure that the model fails to capture. Additionally, the model could improve with regularization, which is the motivation for testing with the addition of L2 regularization with differing lambda (λ) values, or in this case differing C values.

The C values used for hyperparameter tuning of the logistic regression model with L2 regularization were 1000000, 1.0, 0.1, 0.01, 0.001, and 0.0001, in order to see the performance of differing regularization strength on the performance metrics. The following figures show, as previously mentioned, how differing regularization strength influences the performance metrics of the logistic regression model with L2 regularization and scaled untransformed features.

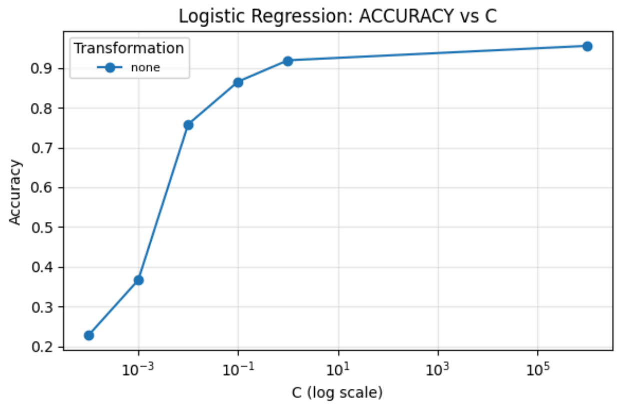

Fig. 6. Accuracy vs C Value for Untransformed Logistic Regression

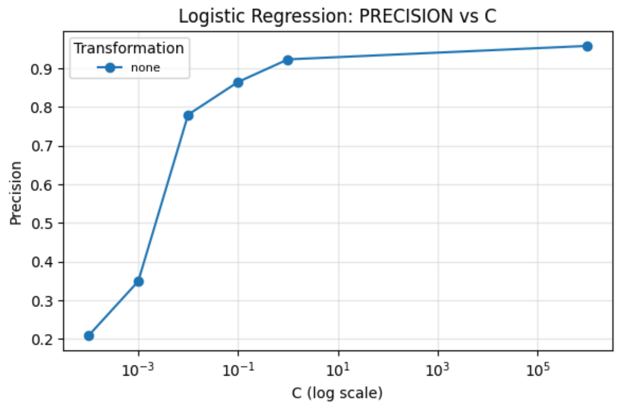

Fig. 7. Precision vs C Value for Untransformed Logistic Regression

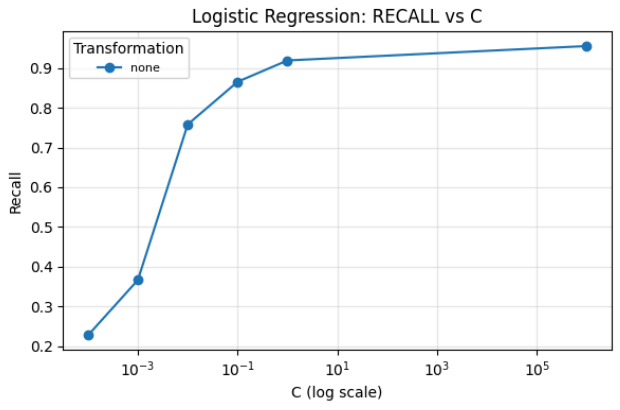

Fig. 8. Recall vs C Value for Untransformed Logistic Regression

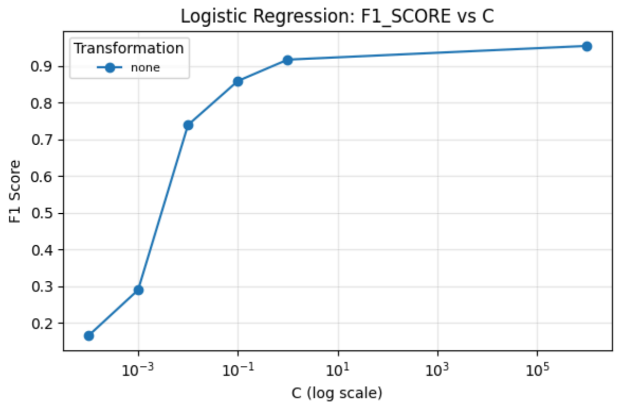

Fig. 9. F1 Score vs C Value for Untransformed Logistic Regression

Across all four metric plots, the performance of the logistic regression model improves as the regularization strength decreases. When a high regularization strength was used, in this case a low C value, the model struggled to perform well and underfit extremely. This led to the use of a lesser regularization strength throughout each hyperparameter tuning. As regularization strength decreased, the model was more flexible to fit the data better. As the regularization strength continued to decrease, the score for each metric plateaus, suggesting that although regularization strength was a factor that influenced how well the model performed, the model could be limited due to the current linear nature of the features. Thus, in the search for a better result and better performance of the logistic regression model, a polynomial of degree two feature transformation was used along with the same hyperparameter tuning values.

For a feature transformation of polynomial degree two, the number of features changes from 11 to 77. The C values used for the hyperparameter tuning remained the same for these features, in order to continue to see the performance of differing regularization strength on the performance metrics. Previously a low regularization strength led to a high performing model. However, this time the issue of overfitting could be possible since there are seven times as many features. The following figures show the metrics vs the C values used for this logistic regression model with polynomial degree two feature transformations.

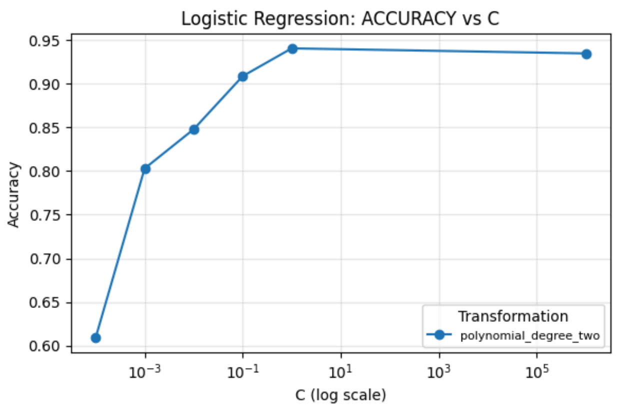

Fig. 10. Accuracy vs C Value for Polynomial Degree Two Feature Transformation Logistic Regression

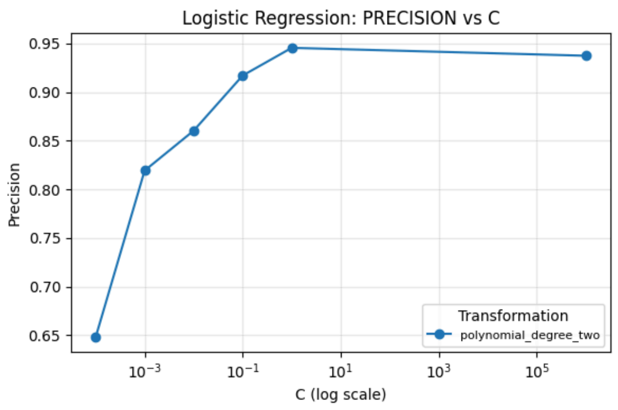

Fig. 10. Precision vs C Value for Polynomial Degree Two Feature Transformation Logistic Regression

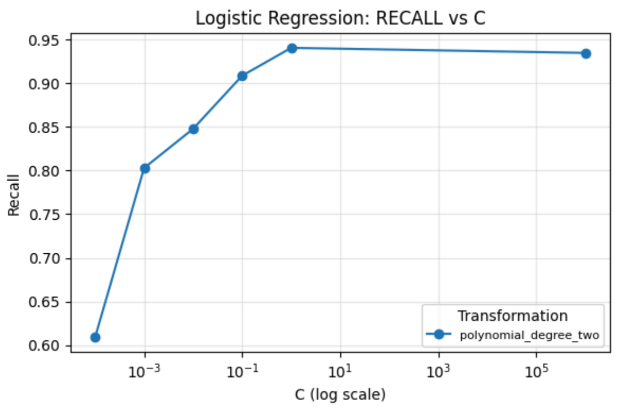

Fig. 12. Recall vs C Value for Polynomial Degree Two Feature Transformation Logistic Regression

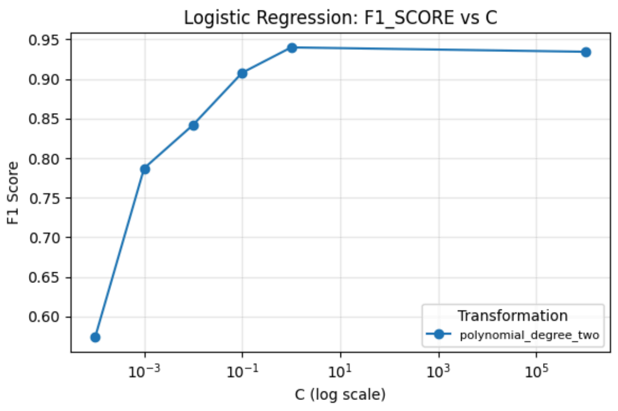

Fig. 13. F1 Score vs C Value for Polynomial Degree Two Feature Transformation Logistic Regression

As seen previously with untransformed features, a high regularization strength limits the model to not perform as well, not as severely as before which can be accredited to the increase in complexity of the model, but it does not perform as wanted. As regularization strength decreased, as previously seen, the score of each metric increased. However, something to notice is that there was a noticeable difference between the training accuracy and the validation accuracy in the models trained. The training accuracy tended to be considerably higher than the validation accuracy, which suggests that the model is starting to overfit due to the number of features. In order to confirm this finding and try to find a balance between regularization strengths, a second feature transformation was applied. A polynomial degree three feature transformation was used.

For a feature transformation of polynomial degree three, the number of features changes from the original 11 to 363. The C values used for the hyperparameter tuning remained the same for these features, in order to continue to see the performance of differing regularization strength on the performance metrics and to continue to confirm previous findings. As previously shown, a low regularization strength led to a high performing model. However, this time the issue of overfitting is present since there are 33 times as many features than the original number of features. Going in, a positive outcome was not expected, but curiosity over more complex relationships in the multi-class classification problem was present. The following figures show the metrics vs the C values used for this logistic regression model with polynomial degree three feature transformations.

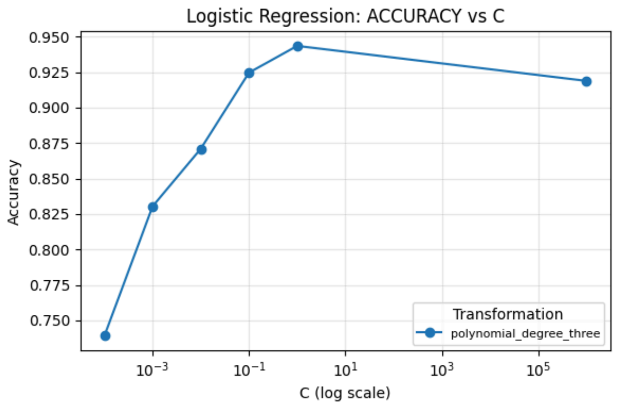

Fig. 14. Accuracy vs C Value for Polynomial Degree Three Feature Transformation Logistic Regression

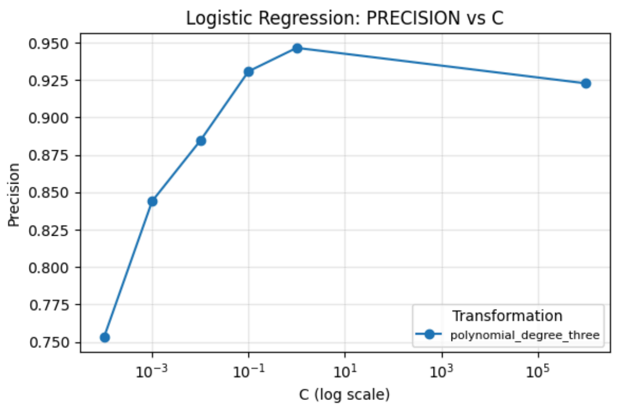

Fig. 15. Precision vs C Value for Polynomial Degree Three Feature Transformation Logistic Regression

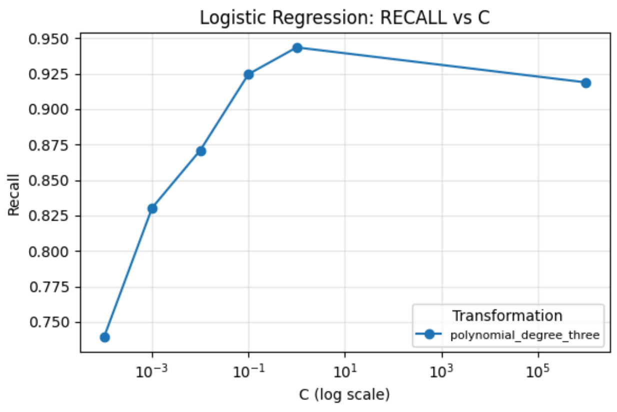

Fig. 16. Recall vs C Value for Polynomial Degree Three Feature Transformation Logistic Regression

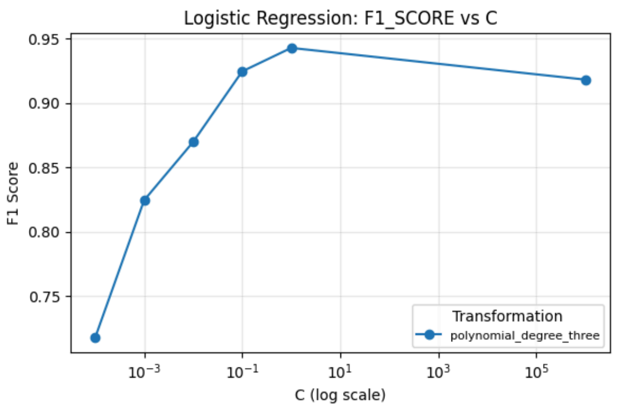

Fig. 17. F1 Score vs C Value for Polynomial Degree Three Feature Transformation Logistic Regression

As previously suspected, a higher degree of polynomial feature transformation did not increase the overall scores or improve the model performance substantially when compared to degree two. While, as seen previously before, the high regularization strength did provide a higher metric score for this feature transformation, the model did not have a higher performance and was more susceptible to memorizing noise and strayed away from learning the true pattern. Thus, the components did not require a high degree of polynomial features and the addition of nonlinearity added unnecessary noise.

Continuous testing with the same hyperparameter tuning values led to the optimal C value to be around 0.1 and 1.0, since it showed the closest values between training and validation accuracy and performed the highest for the F1 score. Which will be carried on in k-means clustering.

K-means clustering was chosen because adding additional polynomial features was evident that it would not improve significantly and that the model would overfit dramatically, perhaps even in the optimal C value range. Thus, through k-means clustering, non-linear distance features could prove to be more useful than an increase in degree of complexity. As previously mentioned, the number of clusters chosen stayed constant at eight clusters. The following figures show the metrics vs the C values used for this logistic regression model with k-means clustering feature transformation.

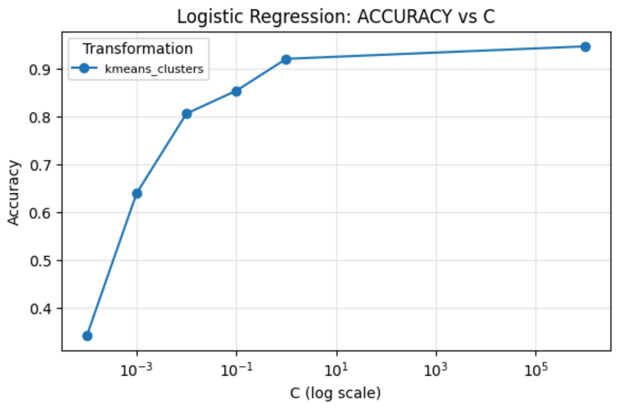

Fig. 18. Accuracy vs C Value for K-Means Clustering Feature Transformation Logistic Regression

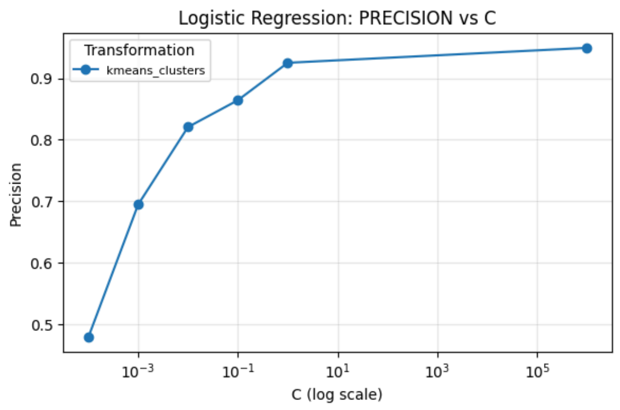

Fig. 19. Precision vs C Value for K-Means Clustering Feature Transformation Logistic Regression

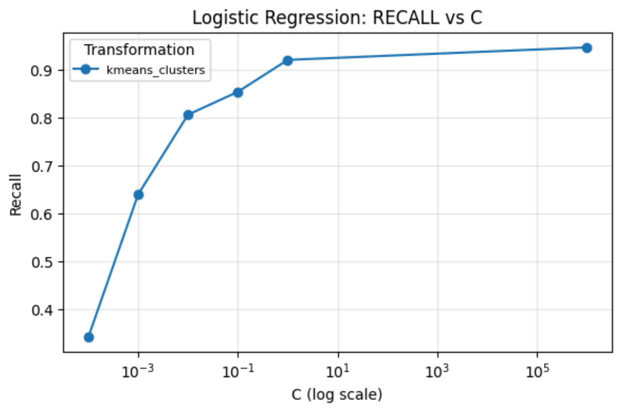

Fig. 20. Recall vs C Value for K-Means Clustering Feature Transformation Logistic Regression

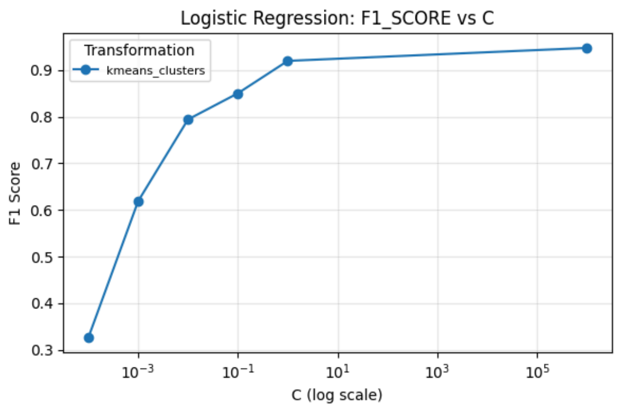

Fig. 21. F1 Score vs C Value for K-Means Clustering Feature Transformation Logistic Regression

With k-means clustering the pattern seen previously is still present in this modeling process and this method of feature transformation did not perform worse than a polynomial feature transformation of degree three, but did not perform better than the previous modeling either. This finding could have been because one value of k (number of clusters) was used and is something to take into consideration in the future.

Thus, each of the metrics and models were saved for a final evaluation and modeling K-Nearest Neighbors (KNN) followed.

B. K-NEAREST NEIGHBORS

For KNN, KNeighborsClassifier() was used from sklearn.neighbors. KNN was chosen because it does not create decision boundaries, but rather it groups points together in feature space based on labels. The baseline KNN model used had a k value of 5 and a weight value of ‘uniform’, which means that all points in each group are weighted equally, in order to get the baseline metrics and a direction to how said metrics could improve. A confusion matrix was produced and showed the following:

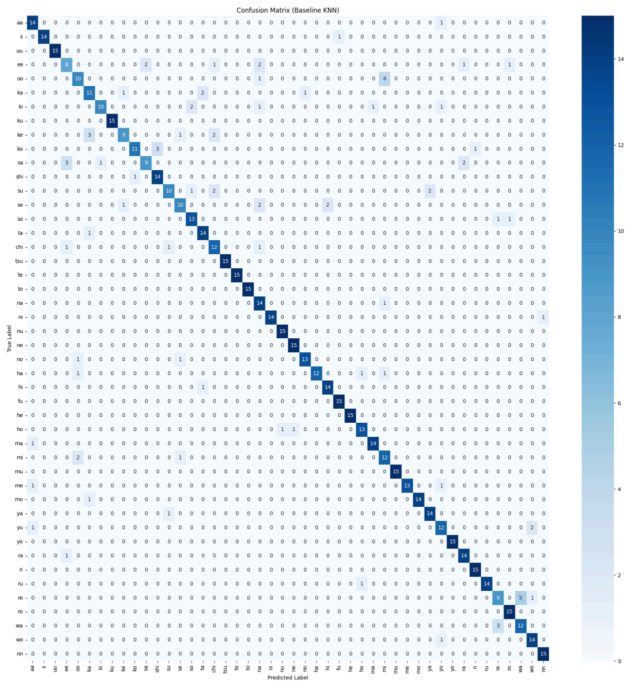

Fig. 22. Confusion Matrix for Baseline KNN Model

Figure 22 showed a relatively high number of examples that were categorized accurately, which is reflected in Table 3.

TABLE III
METRICS FOR BASELINE KNN

Metric	**Score**
Training Accuracy	0.9245
Validation Accuracy	0.8710
Validation Precision	0.8750
Validation Recall	0.8710
Validation F1 Score	0.8693

The baseline model achieves a relatively high performance and suggests decent generalization, however the results also suggest room for improvement. As previously performed, possible non-linear relationships/structure could help KNN since the metrics of the baseline model were not that high when compared to logistic regression. Additionally, the model could improve with different k values along with using ‘distance’, where closer neighbors/points of a point will have a greater influence than neighbors/points which are further away, over ‘uniform’, which is the motivation for testing.

The k values used for hyperparameter tuning were 1, 5, and 10. As previously mentioned and defined, the weight functions used were ‘uniform’ and ‘distance’. These values stayed constant throughout each model testing with different feature transformations in order to see how these hyperparameter values influenced the metric scores. The following figures show how different k values and weight functions influence a KNN model with scaled untransformed features.

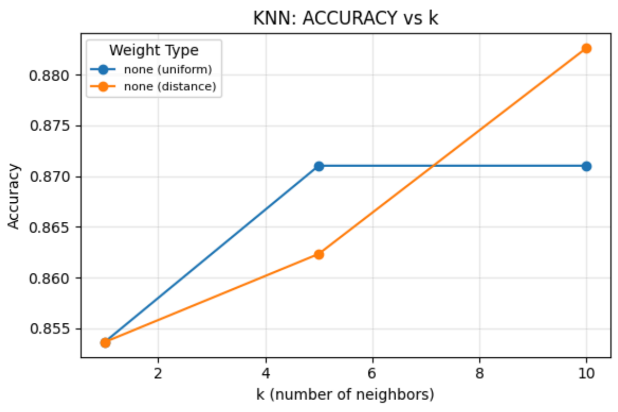

Fig. 23. Accuracy vs K Value for Untransformed KNN

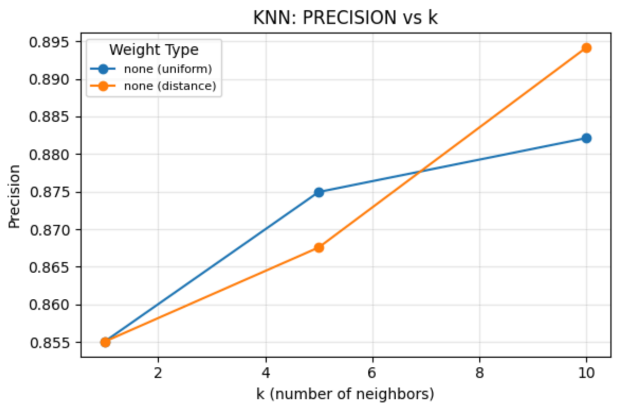

Fig. 24. Precision vs K Value for Untransformed KNN

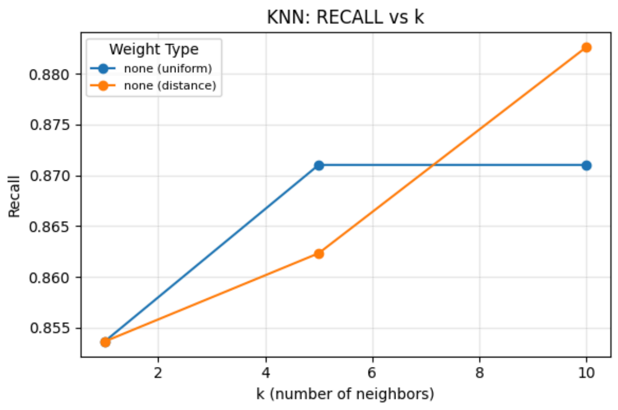

Fig. 25. Recall vs K Value for Untransformed KNN

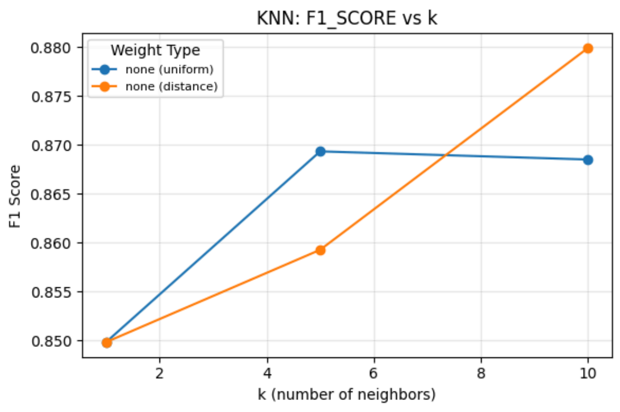

Fig. 26. F1 Score vs K Value for Untransformed KNN

Across all four metrics the general consensus is that higher number of neighbors, increased the overall performance metrics of the model. This makes sense because a lower number of neighbors tends to mean that the model is capturing more noise rather than the true pattern of the multi-class classification problem. Regarding weight functions, it seems that a distance weight function provides a better performance metric as the number of neighbors increases. This makes sense because closer neighbors should influence more than other neighbors because points could be swayed to a wrongful prediction and lower the impact of more significant points. In this modeling training, overfitting was a problem, but lessened as the number of neighbors increased.

In an effort to increase these metrics, a polynomial of degree two feature transformation, as previously done with logistic regression, could positively influence the model and capture nonlinear structure and benefit the KNN modeling. Using the same hyperparameter tuning values, the following figures show how different k values and weight functions influence a KNN model with polynomial of degree two transformed features.

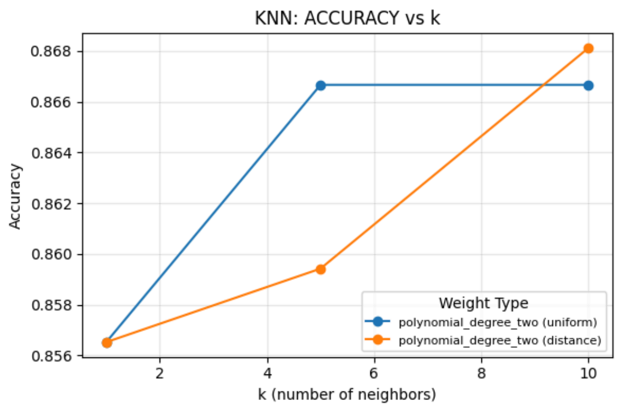

Fig. 27. Accuracy vs K Value for Polynomial Degree Two Feature Transformation KNN

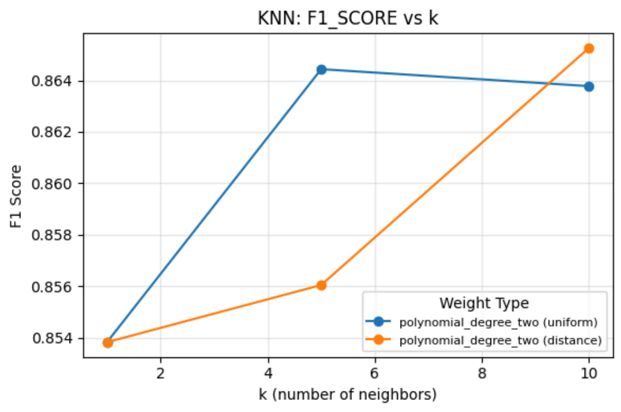

Fig. 30. F1 Score vs K Value for Polynomial Degree Two Feature Transformation KNN

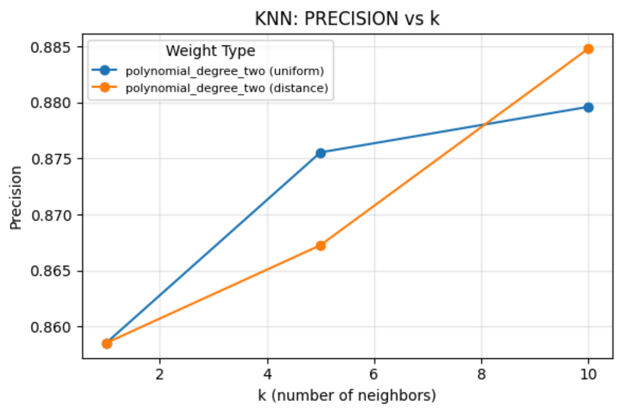

Fig. 28. Precision vs K Value for Polynomial Degree Two Feature Transformation KNN

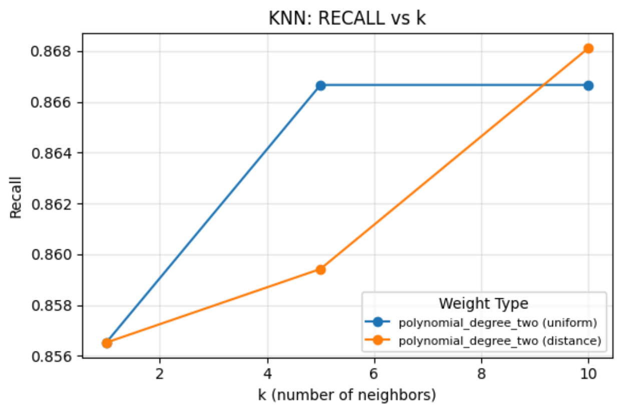

Fig. 29. Recall vs K Value for Polynomial Degree Two Feature Transformation KNN

As seen previously, a higher neighbor count benefits the model's performance metric and a weight function of distance continues to outperform as the number of neighbors increases as well. A polynomial feature transformation seemed to help improve the metrics, but there is a pivotal difference. Overfitting was more of an issue here since the difference between the training accuracy and validation accuracy was significantly higher than before. However, for the sake of continuing to find the experiment on the number of neighbors and weight function type, a polynomial feature transformation of degree three will be applied to the model.

The following figures show how different k values and weight functions influence a KNN model with polynomial of degree three transformed features.

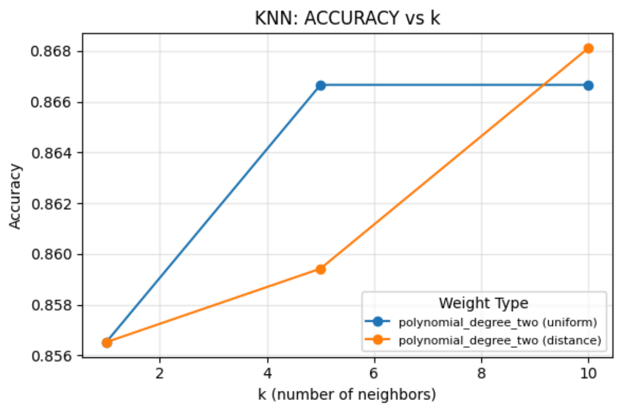

Fig. 31. Accuracy vs K Value for Polynomial Degree Three Feature Transformation KNN

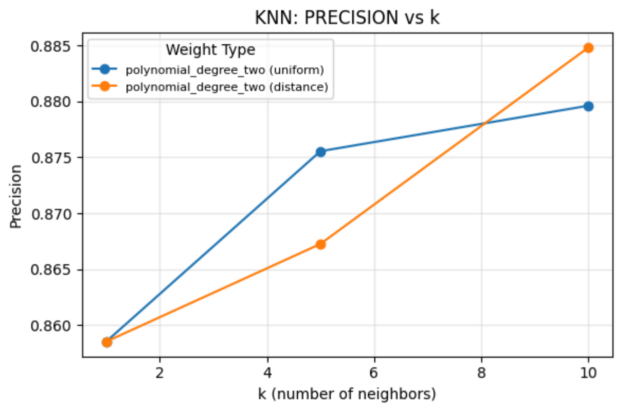

Fig. 32. Precision vs K Value for Polynomial Degree Three Feature Transformation KNN

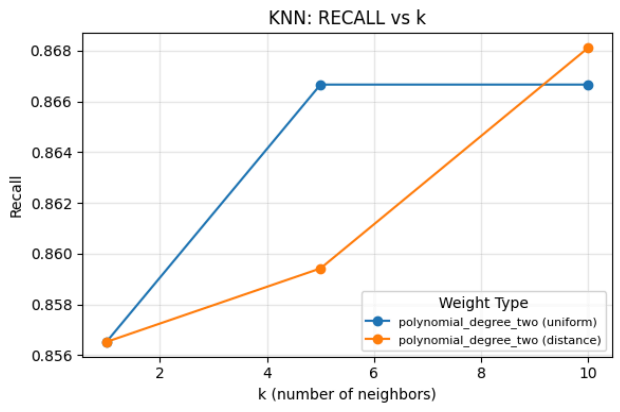

Fig. 33. Recall vs K Value for Polynomial Degree Three Feature Transformation KNN

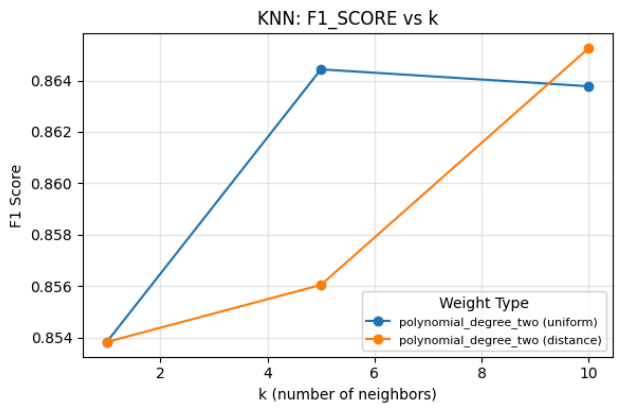

Fig. 34. F1 Score vs K Value for Polynomial Degree Three Feature Transformation KNN

Even though the graphs seem to continue to prove the point that a higher number of neighbors and that a weight function of distance is beneficial, the addition of polynomial features made every experiment here except for a weight function of uniform and neighbor count of 10 overfit drastically. This could be because the number of neighbors was not increased with each feature transformation and the model is incredibly sensitive to noise. Thus, any further polynomial feature transformation would not be useful.

Since the polynomial feature transformation increased the number of features drastically without improving KNN performance, leading to overfitting, k-means clustering was explored next as a way to introduce nonlinearity without increasing the feature space drastically. The following figures show how different k values and weight functions influence a KNN model with k-means clustering transformed features, where k = 8.

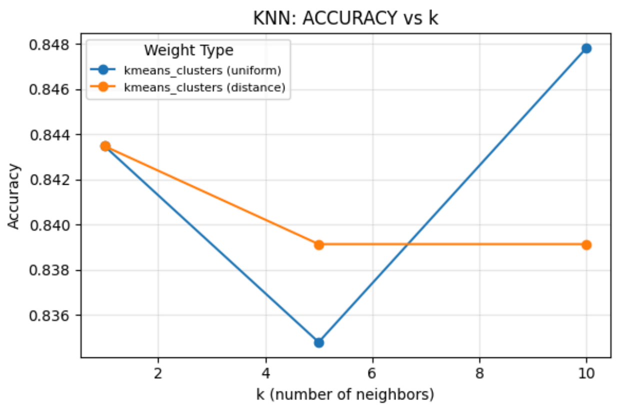

Fig. 35. Accuracy vs K Value for K-Means Clustering Feature Transformation KNN

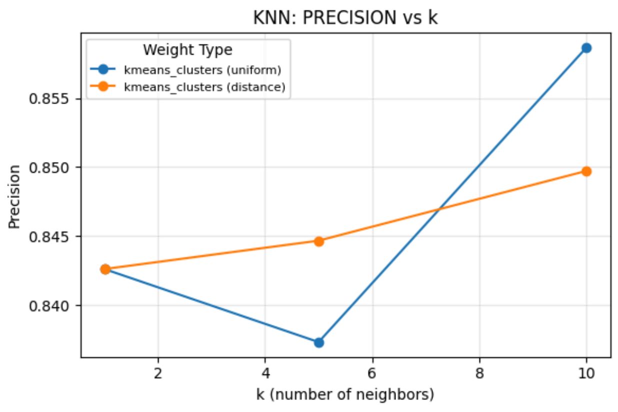

Fig. 36. Precision vs K Value for K-Means Clustering Feature Transformation KNN

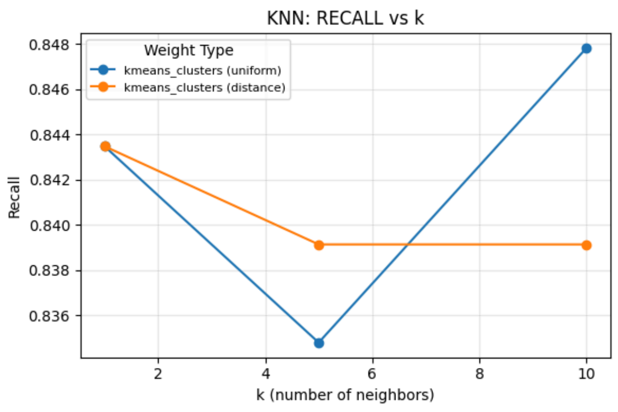

Fig. 37. Recall vs K Value for K-Means Clustering Feature Transformation KNN

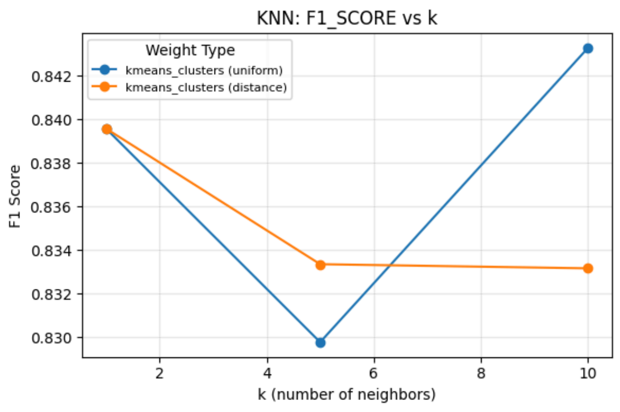

Fig. 38. F1 Score vs K Value for K-Means Clustering Feature Transformation KNN

In this case, uniform weight function tends to lead to a higher model performance and the model did not overfit as much as previous models when the weight function was uniform and the number of neighbors, k, increased. This may be due to the combination of k-means clusters and KNN leading to distance to not be a greater factor as it was previously.

Nevertheless, overall KNN underperformed in comparison to logistic regression, but each metric was saved for a final evaluation and modeling Neural Networks followed.

C. NEURAL NETWORKS

For neural networks, MLPClassifier() from sklearn.neural_network was used. Neural Networks was picked because it has the ability of modeling nonlinear relationships between features and is suitable for classification problems involving visual patterns, such as handwriting variation. The baseline neural network was a one hidden layer network with 64 neurons, with sigmoid activation function, and no regularization, in order to get the baseline metrics and a direction to how said metrics could improve. A confusion matrix was produced and showed the following:

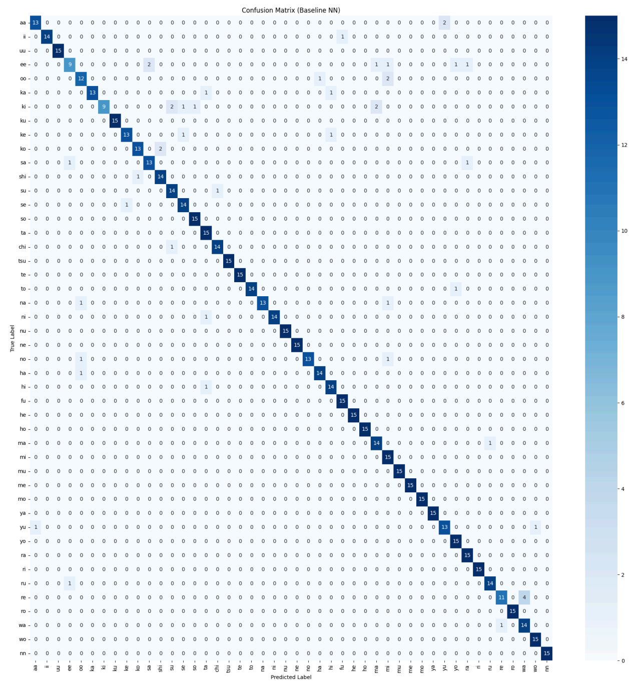

Fig. 39. Confusion Matrix for Baseline Neural Network Model

Figure 39 showed a high number of examples that were categorized accurately, which is reflected in Table 4.

TABLE IV
METRICS FOR BASELINE NEURAL NETWORK

Metric	**Score**
Training Accuracy	0.9531
Validation Accuracy	0.9319
Validation Precision	0.9362
Validation Recall	0.9319
Validation F1 Score	0.9308

The baseline model achieved a high accuracy score in both training and validation, therefore suggesting that overfitting currently is not much of a problem and should be maintainable once L2 regularization is added onto the baseline model. The table also suggests that there is room for improvement. However, instead of transforming features as it did not prove to be that helpful previously, the only type of transformation that occurred was the changing in architecture. Different numbers of hidden layers and neurons per layer will be used as transformation metrics along with hyperparameter tuning. In this case, the hyperparameter tuning were the activation functions and alpha values, L2 regularization strength.

The activation functions used during the hyperparameter tuning were sigmoid, ReLU, and Tanh. The alpha values, directly related to L2 regularization strength, used were 0.000001, 0.0001, 0.001, 0.01, 0.1, and 1.0. These values stayed constant throughout each iteration of neural network architecture. To start, consider a one hidden layer neural network with L2 regularization of various strengths and various activation functions. The following figures show how different hyperparameter values influence a neural network model with scaled untransformed features.

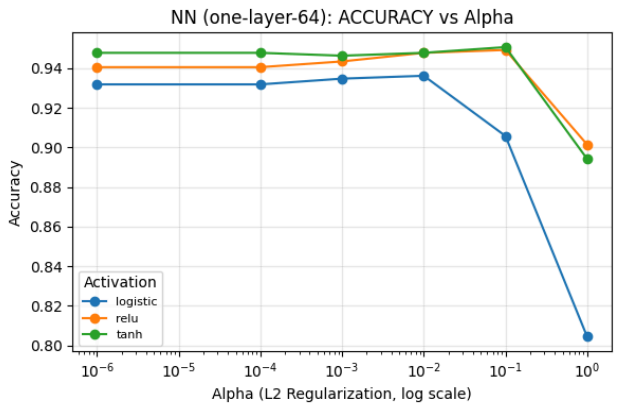

Fig. 40. Accuracy vs Alpha for One Layer Neural Network

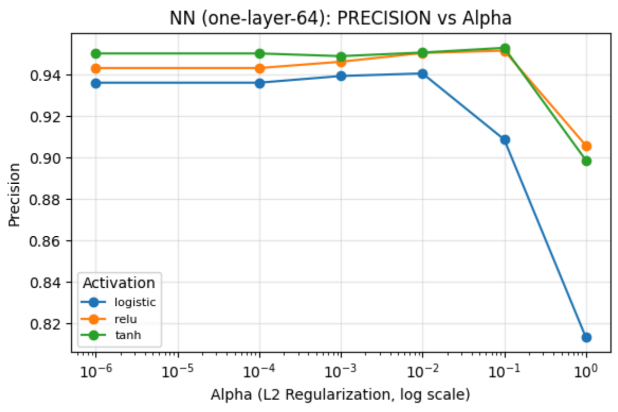

Fig. 41. Accuracy vs Alpha for One Layer Neural Network

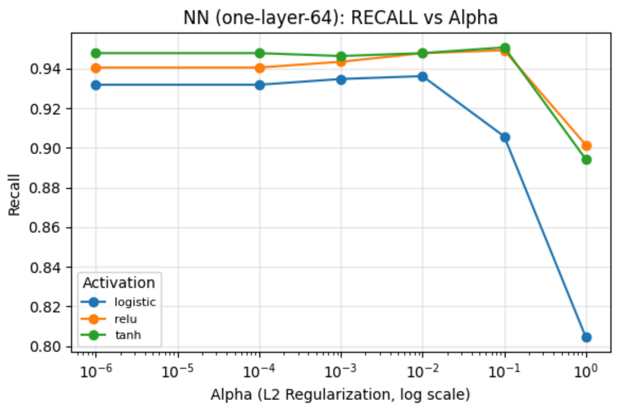

Fig. 42. Recall vs Alpha for One Layer Neural Network

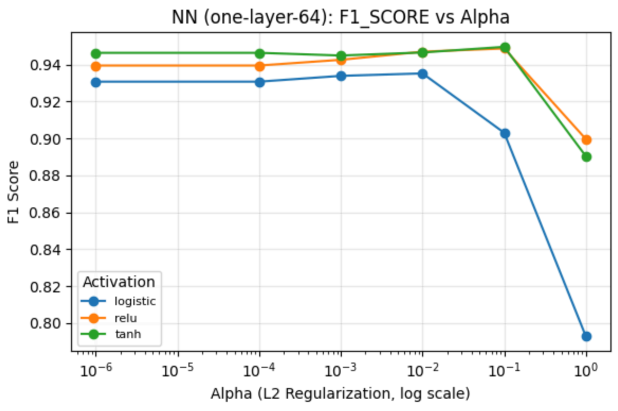

Fig. 43. F1 Score vs Alpha for One Layer Neural Network

As seen from the previous figures, across every activation function and regularization strength, the model consistently achieved a high validation accuracy, precision, recall, and F1 scores, with the best being around 0.94 to 0.95 across most metrics. However, the results plateaued and eventually decreased as regularization increased, showing a limitation with a one layer approach. This pattern suggests that the model, although successful, had reached the linear representative capacity. Additionally, ReLU and tanh performed similarly at around 0.1 as the alpha value, so this is worthy to note. These observations motivated the transformation of a deeper network, which was a two hidden layer neural network with 64 and 32 neurons respectively, allowing the model to improve generalization beyond what a single layer neural network could achieve.

The following figures show how different hyperparameter values influence a two layer neural network model with scaled untransformed features.

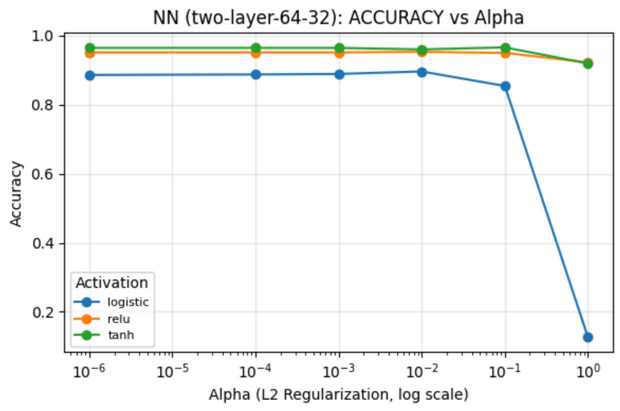

Fig. 44. Accuracy vs Alpha for Two Layer Neural Network

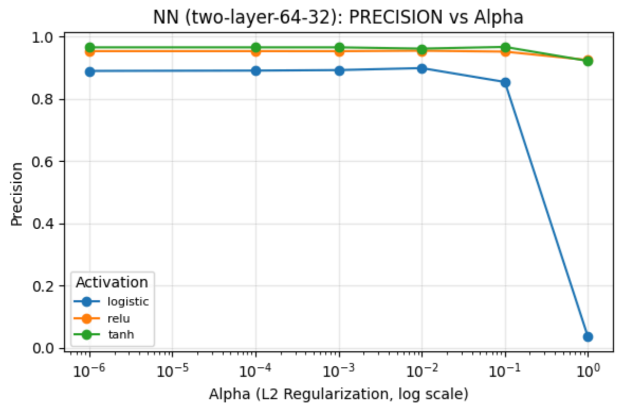

Fig. 45. Precision vs Alpha for Two Layer Neural Network

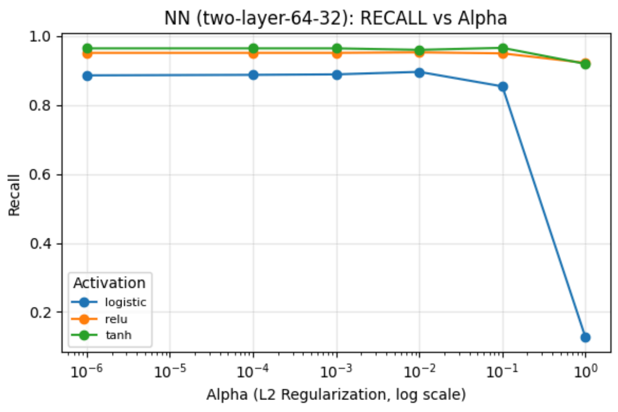

Fig. 46. Recall vs Alpha for Two Layer Neural Network

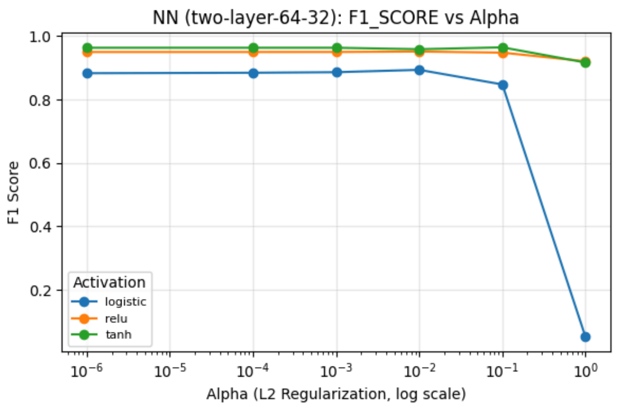

Fig. 47. F1 Score vs Alpha for Two Layer Neural Network

The two layer model showed an improvement over the single hidden layer architecture. The validation accuracy rose approximately to around 0.96 and the F1 scores were about the same, with overfitting not being an issue since training and validation accuracy were close. For tanh and ReLU activations, the best alpha hyperparameter was still around 0.1. These results confirmed that adding more layers allowed the neural network to capture nonlinear relationships that the single hidden layer model could not fully encapsulate. However, despite the improvements, the results plateaued again and increasing alpha still caused a drop. The logistic activation function still underperformed across all values of alpha. This pattern might suggest that the model might still be limited due to the number of hidden layers and that an additional hidden layer could allow the model to learn more. As a result, these observations motivated the transformation of a deeper network, which was a three hidden layer neural network with 64, 64, and 32 neurons respectively, allowing the model to have a chance to improve generalization beyond what a two layer neural network could achieve.

The following figures show how different hyperparameter values influence a three layer neural network model with scaled untransformed features.

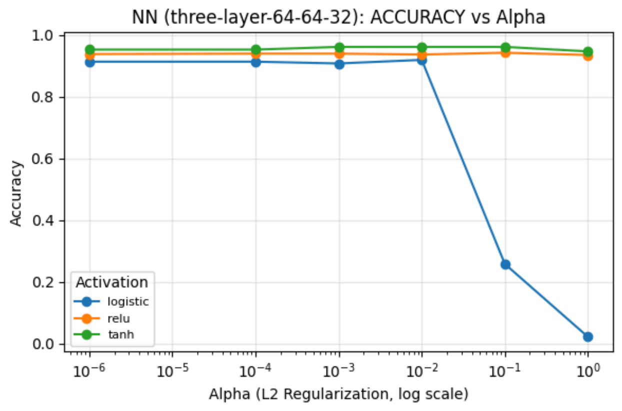

Fig. 48. Accuracy vs Alpha for Three Layer Neural Network

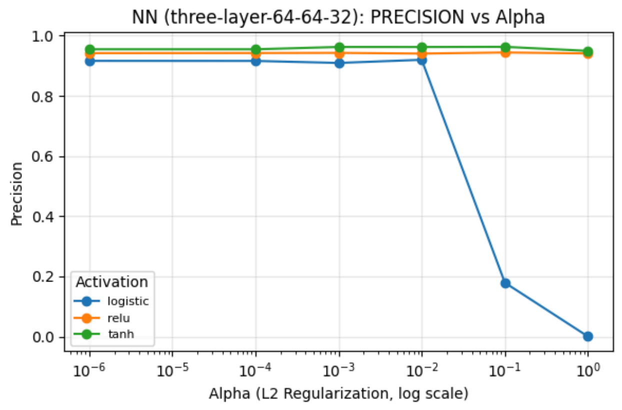

Fig. 49. Precision vs Alpha for Three Layer Neural Network

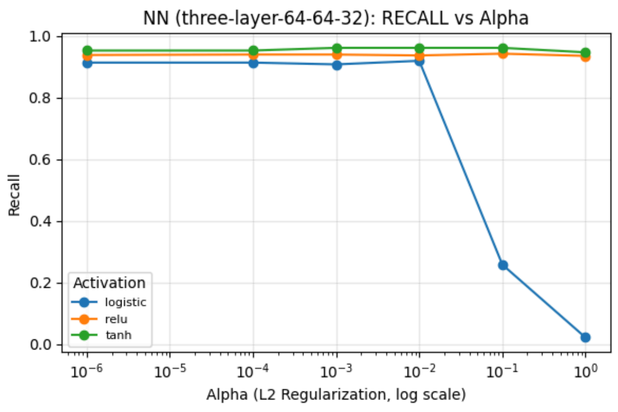

Fig. 50. Recall vs Alpha for Three Layer Neural Network

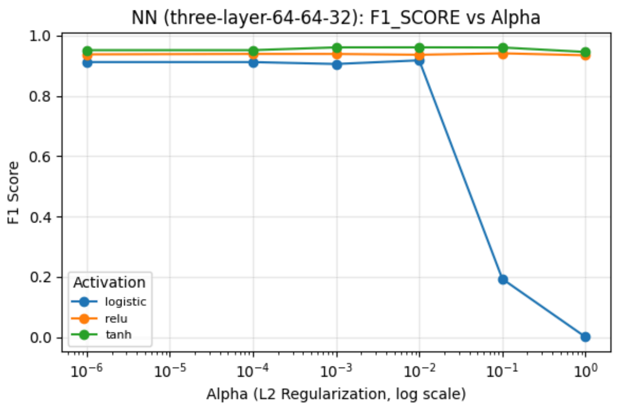

Fig. 51. F1 Score vs Alpha for Three Layer Neural Network

The three layer model demonstrated strong performance as well, but performed a little worse than the previous findings. This confirmed that adding depth allowed the model to learn and improve in some areas, especially in not causing tanh and ReLU activations metrics to drop when alpha increased to 1.0. However, despite these improvements, the performance gains over the two layer network were not drastic. Additionally, there is some evidence of overfitting as training accuracy was reaching 0.99 and the validation accuracy was straying away from the training accuracy. These patterns suggested that depth alone had negative effects and that a neural network model might have done better if the number of neurons in a two layer approach increased. As a result, these observations motivated the transformation of a wider two hidden layer neural network, which was 128 neurons and 64 neurons respectively, allowing the model to have a chance to improve generalization and limit overfitting without the unnecessary depth in the neural network.

The following figures show how different hyperparameter values influence a wider two layer neural network model with scaled untransformed features.

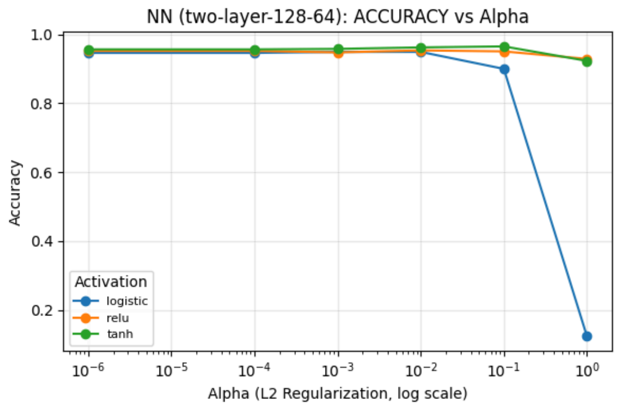

Fig. 52. Accuracy vs Alpha for Wider Two Layer Neural Network

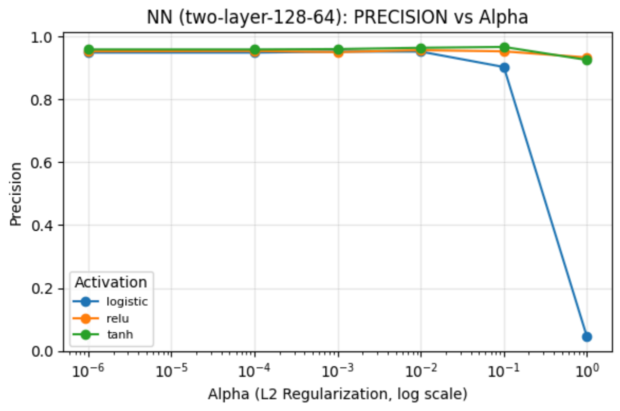

Fig. 53. Precision vs Alpha for Wider Two Layer Neural Network

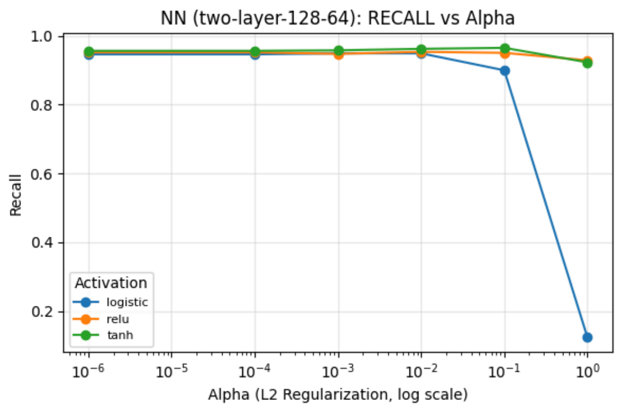

Fig. 54. Recall vs Alpha for Wider Two Layer Neural Network

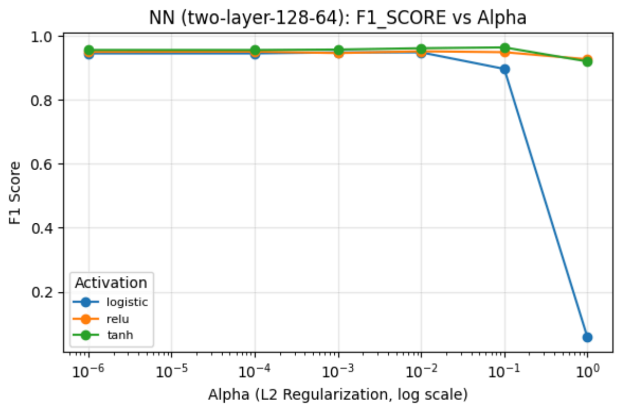

Fig. 55. F1 Score vs Alpha for Wider Two Layer Neural Network

The wider two hidden layer model produced a stronger performance compared to the earlier three hidden layer model and delivered similar results to the best results of previous models. However, there is still an element of overfitting and each metric drops as alpha reaches 1.0. Even though this model was more effective than the three hidden layer network, a wider neural network still had its issues. Nevertheless, these results confirm that, for this dataset and feature space, widening layers rather than deepening the neural network proved to be more effective. Each metric was saved for a final evaluation and ready to be displayed.

Therefore, after every model with or without feature transformation was run with varying hyperparameter tuning, a decision was made on which model to run the test set on.

IV. TABLE OF RESULTS

After careful consideration of the previous figures and results from every model trained, a model was picked based on the overall metrics from the following tables.

TABLE V

NO FEATURE TRANSFORMATION LOGISTIC REGRESSION RESULTS

Transformation	C	Train Accuracy	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
None	1000000	0.980124	0.955072	0.958045	0.955072	0.954616
None	1	0.926708	0.918841	0.923226	0.918841	0.917269
None	0.1	0.874534	0.865217	0.864772	0.865217	0.859132
None	0.01	0.769565	0.757971	0.780226	0.757971	0.739163
None	0.001	0.378261	0.366667	0.349217	0.366667	0.289839
None	0.0001	0.22764	0.227536	0.207941	0.227536	0.164634

TABLE VI

POLYNOMIAL DEGREE ONE FEATURE TRANSFORMATION LOGISTIC REGRESSION RESULTS

Transformation	C	Train Accuracy	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
Polynomial Degree Two	1000000	0.999379	0.934783	0.937291	0.934783	0.934437
Polynomial Degree Two	1	0.964907	0.94058	0.945412	0.94058	0.939949
Polynomial Degree Two	0.1	0.926708	0.908696	0.916955	0.908696	0.907937
Polynomial Degree Two	0.01	0.868634	0.847826	0.860262	0.847826	0.842087
Polynomial Degree Two	0.001	0.804658	0.802899	0.81924	0.802899	0.78721
Polynomial Degree Two	0.0001	0.637578	0.608696	0.647703	0.608696	0.573405

TABLE VII
POLYNOMIAL DEGREE THREE FEATURE TRANSFORMATION LOGISTIC REGRESSION RESULTS

Transformation	C	Train Accuracy	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
Polynomial Degree Three	1000000	1	0.918841	0.922812	0.918841	0.91823
Polynomial Degree Three	1	0.986335	0.943478	0.946543	0.943478	0.942893
Polynomial Degree Three	0.1	0.954969	0.924638	0.930821	0.924638	0.924619

Polynomial Degree Three	0.01	0.896273	0.871014	0.884496	0.871014	0.869773
Polynomial Degree Three	0.001	0.850621	0.830435	0.844011	0.830435	0.824409
Polynomial Degree Three	0.0001	0.759317	0.73913	0.752908	0.73913	0.71737

TABLE VIII

K-MEANS CLUSTERS FEATURE TRANSFORMATION LOGISTIC REGRESSION RESULTS

Transformation	C	Train Accuracy	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
K-Means Clusters	1000000	0.98913	0.947826	0.949213	0.947826	0.947447
K-Means Clusters	1	0.932609	0.921739	0.924966	0.921739	0.919641
K-Means Clusters	0.1	0.875466	0.855072	0.86423	0.855072	0.850169
K-Means Clusters	0.01	0.811801	0.807246	0.820718	0.807246	0.794112
K-Means Clusters	0.001	0.662422	0.64058	0.694914	0.64058	0.618856
K-Means Clusters	0.0001	0.336025	0.342029	0.479296	0.342029	0.325889

TABLE IX

NO FEATURE TRANSFORMATION KNN RESULTS

Transformation	Weight Functions	k	Train Accuracy	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
None	uniform	1	1	0.853623	0.855039	0.853623	0.849797
None	uniform	5	0.924534	0.871014	0.874953	0.871014	0.869326
None	uniform	10	0.9	0.871014	0.882125	0.871014	0.868495
None	distance	1	1	0.853623	0.855039	0.853623	0.849797
None	distance	5	1	0.862319	0.867572	0.862319	0.859252
None	distance	10	1	0.882609	0.894165	0.882609	0.8799

TABLE X

POLYNOMIAL DEGREE TWO FEATURE TRANSFORMATION KNN RESULTS

Transformation	Weight Functions	k	Train Accuracy	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
Polynomial Degree Two	uniform	1	1	0.856522	0.8585	0.856522	0.853823
Polynomial	uniform	5	0.91677	0.866667	0.875551	0.866667	0.864432

Degree Two							
Polynomial Degree Two	uniform	10	0.892857	0.866667	0.879621	0.866667	0.863775
Polynomial Degree Two	distance	1	1	0.856522	0.8585	0.856522	0.853823
Polynomial Degree Two	distance	5	1	0.85942	0.867249	0.85942	0.856044
Polynomial Degree Two	distance	10	1	0.868116	0.884826	0.868116	0.865253

TABLE XI

POLYNOMIAL DEGREE THREE FEATURE TRANSFORMATION KNN RESULTS

Transformation	Weight Functions	k	Train Accuracy	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
Polynomial Degree Three	uniform	1	1	0.857971	0.863809	0.857971	0.856219
Polynomial Degree Three	uniform	5	0.909938	0.869565	0.879138	0.869565	0.868049
Polynomial Degree Three	uniform	10	0.882298	0.850725	0.869675	0.850725	0.849602
Polynomial Degree Three	distance	1	1	0.857971	0.863809	0.857971	0.856219
Polynomial Degree Three	distance	5	1	0.855072	0.863118	0.855072	0.851195
Polynomial Degree Three	distance	10	1	0.856522	0.878217	0.856522	0.854744

TABLE XII

K-MEANS CLUSTERS FEATURE TRANSFORMATION KNN RESULTS

Transformation	Weight Functions	k	Train Accuracy	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
K-Means Clusters	uniform	1	1	0.843478	0.842618	0.843478	0.83958
K-Means Clusters	uniform	5	0.903416	0.834783	0.83733	0.834783	0.829771
K-Means Clusters	uniform	10	0.871739	0.847826	0.858652	0.847826	0.843278
K-Means Clusters	distance	1	1	0.843478	0.842618	0.843478	0.83958
K-Means Clusters	distance	5	1	0.83913	0.844673	0.83913	0.833352
K-Means Clusters	distance	10	1	0.83913	0.849723	0.83913	0.833162

TABLE XIII

ONE HIDDEN LAYER NEURAL NETWORKS RESULTS

Transformation	Activation Function	Alpha	Train Accuracy	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
One Layer (64)	ReLU	0.000001	0.986335	0.94058	0.943247	0.94058	0.939496
One Layer (64)	ReLU	0.0001	0.986646	0.94058	0.943247	0.94058	0.939496
One Layer (64)	ReLU	0.001	0.986646	0.943478	0.946291	0.943478	0.942598
One Layer (64)	ReLU	0.01	0.98354	0.947826	0.950522	0.947826	0.946991
One Layer (64)	ReLU	0.1	0.963354	0.949275	0.951652	0.949275	0.948739
One Layer (64)	ReLU	1	0.909317	0.901449	0.905869	0.901449	0.899623
One Layer (64)	Tanh	0.000001	0.98323	0.947826	0.950292	0.947826	0.946372
One Layer (64)	Tanh	0.0001	0.98323	0.947826	0.950292	0.947826	0.946372
One Layer (64)	Tanh	0.001	0.982919	0.946377	0.948989	0.946377	0.944915
One Layer (64)	Tanh	0.01	0.980124	0.947826	0.950732	0.947826	0.946561
One Layer (64)	Tanh	0.1	0.961491	0.950725	0.952989	0.950725	0.949613
One Layer (64)	Tanh	1	0.901242	0.894203	0.898759	0.894203	0.890353
One Layer (64)	Sigmoid	0.000001	0.953106	0.931884	0.93618	0.931884	0.930761
One Layer (64)	Sigmoid	0.0001	0.953106	0.931884	0.93618	0.931884	0.930761
One Layer (64)	Sigmoid	0.001	0.952795	0.934783	0.939408	0.934783	0.933899
One Layer (64)	Sigmoid	0.01	0.948447	0.936232	0.940653	0.936232	0.935228
One Layer (64)	Sigmoid	0.1	0.924224	0.905797	0.908772	0.905797	0.903071
One Layer (64)	Sigmoid	1	0.809006	0.804348	0.81325	0.804348	0.792737

TABLE XIV

TWO HIDDEN LAYERS NEURAL NETWORKS RESULTS

Transformation	Activation Function	Alpha	Train Accuracy	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
Two Layer (64, 32)	ReLU	0.000001	0.988199	0.950725	0.952962	0.950725	0.94996
Two Layer (64, 32)	ReLU	0.0001	0.987888	0.950725	0.952962	0.950725	0.94996
Two Layer (64, 32)	ReLU	0.001	0.98913	0.950725	0.95276	0.950725	0.950036
Two Layer (64, 32)	ReLU	0.01	0.987888	0.952174	0.954334	0.952174	0.951414
Two Layer (64, 32)	ReLU	0.1	0.981056	0.949275	0.951446	0.949275	0.947964
Two Layer (64, 32)	ReLU	1	0.936025	0.921739	0.92448	0.921739	0.920815
Two Layer (64, 32)	Tanh	0.000001	0.997205	0.963768	0.965171	0.963768	0.963362
Two Layer (64, 32)	Tanh	0.0001	0.997205	0.963768	0.965171	0.963768	0.963362

Two Layer (64, 32)	Tanh	0.001	0.996894	0.963768	0.965171	0.963768	0.963362
Two Layer (64, 32)	Tanh	0.01	0.995652	0.95942	0.961034	0.95942	0.958744
Two Layer (64, 32)	Tanh	0.1	0.979814	0.965217	0.96656	0.965217	0.964652
Two Layer (64, 32)	Tanh	1	0.925466	0.918841	0.921406	0.918841	0.916785
Two Layer (64, 32)	Sigmoid	0.000001	0.934783	0.885507	0.889376	0.885507	0.883146
Two Layer (64, 32)	Sigmoid	0.0001	0.934783	0.886957	0.890442	0.886957	0.884514
Two Layer (64, 32)	Sigmoid	0.001	0.933851	0.888406	0.892117	0.888406	0.886316
Two Layer (64, 32)	Sigmoid	0.01	0.928571	0.895652	0.898502	0.895652	0.893689
Two Layer (64, 32)	Sigmoid	0.1	0.891304	0.853623	0.853945	0.853623	0.84765
Two Layer (64, 32)	Sigmoid	1	0.124224	0.126087	0.034963	0.126087	0.05123

TABLE XV

THREE HIDDEN LAYERS NEURAL NETWORKS RESULTS

Transformation	Activation Function	Alpha	Train Accuracy	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
Three Layer (64, 64, 32)	ReLU	0.000001	0.992236	0.937681	0.941793	0.937681	0.937327
Three Layer (64, 64, 32)	ReLU	0.0001	0.991615	0.93913	0.942214	0.93913	0.938739
Three Layer (64, 64, 32)	ReLU	0.001	0.992857	0.93913	0.942748	0.93913	0.93863
Three Layer (64, 64, 32)	ReLU	0.01	0.990683	0.936232	0.940376	0.936232	0.93601
Three Layer (64, 64, 32)	ReLU	0.1	0.982298	0.942029	0.944086	0.942029	0.940806
Three Layer (64, 64, 32)	ReLU	1	0.941304	0.934783	0.940878	0.934783	0.934294
Three Layer (64, 64, 32)	Tanh	0.000001	0.998447	0.952174	0.95465	0.952174	0.951269
Three Layer (64, 64, 32)	Tanh	0.0001	0.998447	0.952174	0.95465	0.952174	0.951269
Three Layer (64, 64, 32)	Tanh	0.001	0.999379	0.96087	0.962336	0.96087	0.960466
Three Layer (64, 64, 32)	Tanh	0.01	0.995963	0.96087	0.962235	0.96087	0.960567
Three Layer (64, 64, 32)	Tanh	0.1	0.986957	0.96087	0.962751	0.96087	0.960112
Three Layer (64, 64, 32)	Tanh	1	0.942547	0.946377	0.949353	0.946377	0.945243

Three Layer (64, 64, 32)	Sigmoid	0.000001	0.946894	0.913043	0.916161	0.913043	0.911786
Three Layer (64, 64, 32)	Sigmoid	0.0001	0.946894	0.913043	0.915969	0.913043	0.911747
Three Layer (64, 64, 32)	Sigmoid	0.001	0.94472	0.907246	0.909266	0.907246	0.905439
Three Layer (64, 64, 32)	Sigmoid	0.01	0.92795	0.918841	0.91974	0.918841	0.917489
Three Layer (64, 64, 32)	Sigmoid	0.1	0.265528	0.257971	0.179321	0.257971	0.193225
Three Layer (64, 64, 32)	Sigmoid	1	0.021739	0.021739	0.000473	0.021739	0.000925

TABLE XVI

WIDE TWO HIDDEN LAYERS NEURAL NETWORKS RESULTS

Transformation	Activation Function	Alpha	Train Accuracy	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
Two Layer (128, 64)	ReLU	0.000001	0.98913	0.952174	0.953045	0.952174	0.950551
Two Layer (128, 64)	ReLU	0.0001	0.990994	0.952174	0.95404	0.952174	0.951288
Two Layer (128, 64)	ReLU	0.001	0.990373	0.947826	0.950667	0.947826	0.94683
Two Layer (128, 64)	ReLU	0.01	0.992236	0.953623	0.956711	0.953623	0.952337
Two Layer (128, 64)	ReLU	0.1	0.982298	0.950725	0.952892	0.950725	0.949519
Two Layer (128, 64)	ReLU	1	0.932919	0.928986	0.933577	0.928986	0.927559
Two Layer (128, 64)	Tanh	0.000001	1	0.956522	0.958973	0.956522	0.956384
Two Layer (128, 64)	Tanh	0.0001	1	0.956522	0.958973	0.956522	0.956384
Two Layer (128, 64)	Tanh	0.001	1	0.957971	0.960198	0.957971	0.957746
Two Layer (128, 64)	Tanh	0.01	0.999379	0.962319	0.964295	0.962319	0.961967
Two Layer (128, 64)	Tanh	0.1	0.980124	0.965217	0.967055	0.965217	0.964512
Two Layer	Tanh	1	0.924845	0.923188	0.925652	0.923188	0.920593

(128, 64)							
Two Layer (128, 64)	Sigmoid	0.000001	0.974534	0.946377	0.94908	0.946377	0.945792
Two Layer (128, 64)	Sigmoid	0.0001	0.974534	0.946377	0.94908	0.946377	0.945792
Two Layer (128, 64)	Sigmoid	0.001	0.973913	0.949275	0.952156	0.949275	0.948913
Two Layer (128, 64)	Sigmoid	0.01	0.968323	0.949275	0.952147	0.949275	0.949006
Two Layer (128, 64)	Sigmoid	0.1	0.913665	0.9	0.903063	0.9	0.897255
Two Layer (128, 64)	Sigmoid	1	0.12795	0.123188	0.044455	0.123188	0.057027

V. TEST SET

After choosing the best model after hyperparameter tuning to be the neural network model with two layers, 64 neurons and 32 neurons respectively, with tanh activation function and an alpha value of 0.1, a confusion matrix along with performance metrics were created.

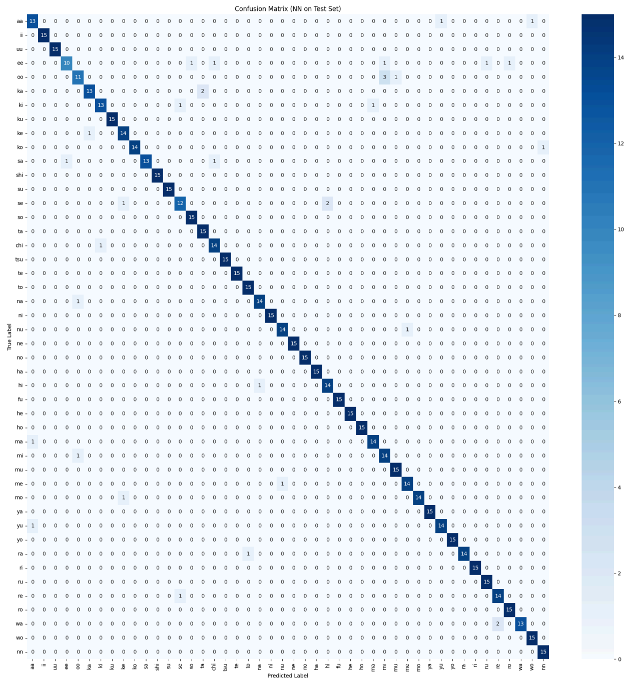

Fig. 56. Confusion Matrix for Best Model

Figure 56 showed a high number of examples that were categorized accurately, which is reflected in Table 17.

TABLE IV
METRICS FOR BEST MODEL

Metric	**Score**
Validation Accuracy	0.9493
Validation Precision	0.9506
Validation Recall	0.9493
Validation F1 Score	0.9486

The table and figure above show a high prediction accuracy and F1 score, leading to a satisfactory result and generalization for the best performing model on unseen data.

VI. CONCLUSION & ANALYTICAL DISCUSSION

In general, across all models feature transformations and regularization played an important role in the key findings of this project.

Regarding logistic regression, feature transformation to a certain extent helped the model encapsulate the nonlinearity that this multi-class classification problem had. When polynomial degree two feature transformation was applied, it helped increase the performance metrics of the model, along with a weaker regularization since it allowed the model to not underfit and restrict itself. However, an important thing to note is that during the polynomial feature transformation for logistic regression, along with other models, overfitting was a recurring issue if the regularization factor was not enough to keep it in check. Polynomial degree two feature transformation was acceptable for each model, whereas polynomial degree three feature transformation captured too much noise and the model was overfitting.

Overall, logistic regression tend to overfit the data if the degree of polynomial features increase and if the C was not large enough, which was evident when $C = 1000000$. Even though these metrics were highest, there was an element of overfitting that balanced out as the value of C increased, lowering the gap between the training accuracy and validation accuracy, but in turn decreased other metrics. Additionally, regarding, bias and variance affected performance of the model because, as previously mentioned, high bias with small C values caused for models to underfit and there to not be meaningful results, whereas high variance was present especially in high degree polynomial features. Thus there needed to be a balance between such.

In the future, training a logistic regression model for a problem like this required perhaps a different approach to the features. Perhaps applying PCA before polynomial feature transformation would control the variance while encapsulating nonlinearity of the problem.

Regarding KNN, feature extraction played an important role in determining the performance of the models. With KNN, the models suffered when polynomial features were used as the models struggled with overfitting the most out of the other models and failed to have high performance metrics like the other models as well.

Overall, as previously mentioned KNN performed relatively well when using scaled untransformed features, but tended to overfit and needed some form of regularization to control it and make meaningful progress with hyperparameter tuning. The hyperparameter of a high number of neighbors and using the distance weight function helped push KNN modeling towards the right direction, but without regularization, KNN failed to succeed.

Thus, in the future, training a KNN requires a higher number of neighbors and regularization in order to make meaningful progress. Additionally, just like logistic regression, applying PCA before polynomial feature transformation would control the variance while encapsulating nonlinearity of the problem, since there high variance was present throughout the modeling.

Lastly, regarding neural networks, increasing the number of layers and neurons per layer made meaningful progress towards finding optimal hyperparameters for the model, along with the various alpha values. Neural network models neither overfit or underfit the data. The model only showed signs of overfitting and underfitting when the number of hidden layers increased to three, otherwise it was moderately well fit. When encountering bias and variance, there was not much bias or variance as the layers in the neural network allowed for flexibility and stable performance metrics. If anything, there were issues with the sigmoid activation as it caused the model to underperform constantly as the alpha value approached 1.0. Thus, having different activation functions and low alpha values allowed the models to not underfit or overfit and perform well.

In the future, with neural networks, focusing on Tanh and ReLU would have been more meaningful as activation functions along with focusing on higher regularization values to see the behavior. Additionally, experimenting with the number of neurons and features chosen could have made the performance and generalization better overall.

Thus from the highlighted key findings in section (IV), those hyperparameters, as previously explained, helped the model perform the best in this project and the neural network was ultimately chosen to have unseen data be exposed to it in order to assess its ability to generalize. Therefore, neural networks seemed to outperform logistic regression, and logistic regression outperformed KNN for this dataset.

REFERENCES

[1] O. F. Beygo, "https://www.kaggle.com/datasets/farukece/handwritten-japanese-hiragana-characters/data," Kaggle.com, Aug. 2025. https://www.kaggle.com/datasets/farukece/handwritten-japanese-hiragana-characters/data