Northeastern

College of Professional Studies
Northeastern University San Jose

MPS Analytics

Course: ALY6020

Assignment:

Module 5 – Project

Submitted on:

October 27, 2023

Submitted to:

Professor: Ahmadi Behzad

Submitted by:

Heejae Roh

Introduction

In this analysis, I will build a KNN model to create a Digit Recognizer. I will examine the parameters in the process of using KNN. After building KNN, I will study the necessary parameters while building a neural network model to predict handwriting. Finally, I will compare the two models and choose the final model.

Analysis

1. Build a KNN model to predict handwriting correctly. Discuss the accuracy that you found and some of the challenges involved with using KNN for this model.

Data preparation and normalization

1. I tried to visualize the image, but failed because some things were not in the pixels. Since the pixels are not composed continuously, it appears to be data in which only some pixels are extracted from the total number of pixels.
2. I used sklearn's train_test_split and set test_size to 0.2. one of the most commonly used random_states, which is 0, was selected. The trained model will be verified through testset.
3. I performed feature scaling using MinMaxScaler. For each value in a feature, MinMaxScaler subtracts the minimum value in the feature and then divides it by the range. The default range for the feature returned by MinMaxScaler is 0 to 1 (Hale, 2019). It helps in preventing features with larger magnitudes from dominating the distance calculations (Filho, 2023). After looking at the data, I decided to choose MinMaxScaler, understanding that pixels contain numbers from 1 to 255 in grayscale to represent an image.

Finding parameters for KNN

```
param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'algorithm': ['ball_tree', 'kd_tree', 'brute'],
    'p': [1, 2]}
```

1. To set the parameters, I first set param_grid and checked the KNN accuracy for various parameters based on this.

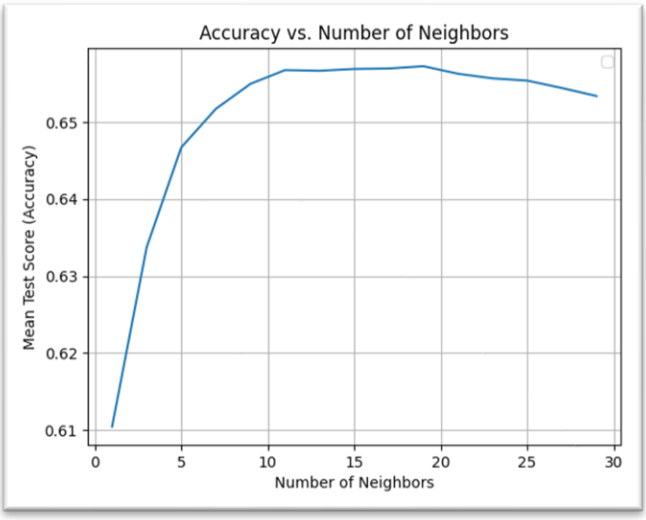
	param_algorithm	param_n_neighbors	param_p	param_weights	mean_test_score	std_test_score	rank_test_score
47	brute	9	2	distance	0.649464	0.003665	1
31	kd_tree	9	2	distance	0.649375	0.003135	2
15	ball_tree	9	2	distance	0.649375	0.003135	2
43	brute	7	2	distance	0.648006	0.005553	4
46	brute	9	2	uniform	0.647589	0.003437	5

2. I decided that the number of points should be around 9, and in this case, I chose distance because I thought it would be right to determine the weight by considering the distance of each point. 'p' means 'different distance metrics are available which use for the tree (like KDTree or Ball Tree)'. If p = 1, this is equivalent to using manhattan_distance(l_1). If p = 2, this is equivalent to using euclidean_distance(l_2). If p > 2, It is minkowski distance(l_p) (GÜNAY, 2020).

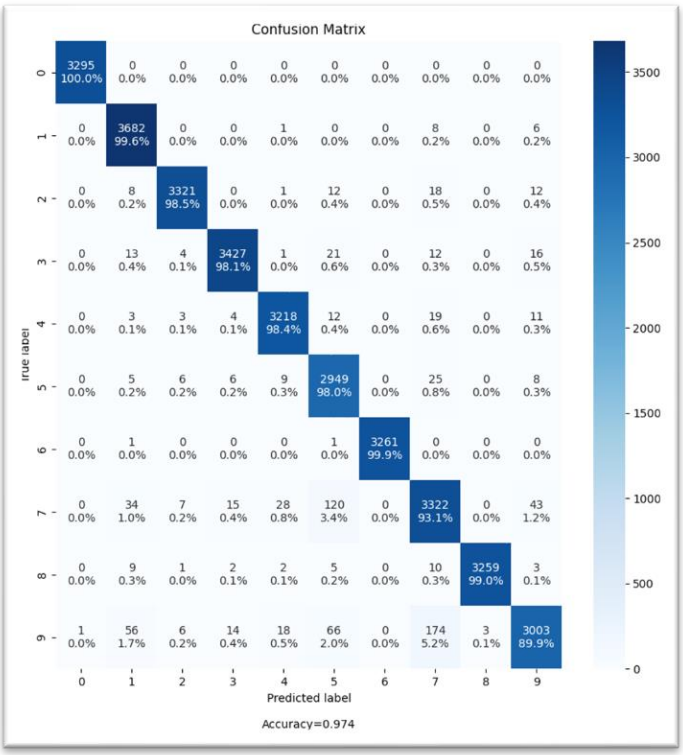
```
param_grid = {'n_neighbors': [9], 'weights': ['distance'],
'algorithm': ['brute'], 'p': [2]
}
```

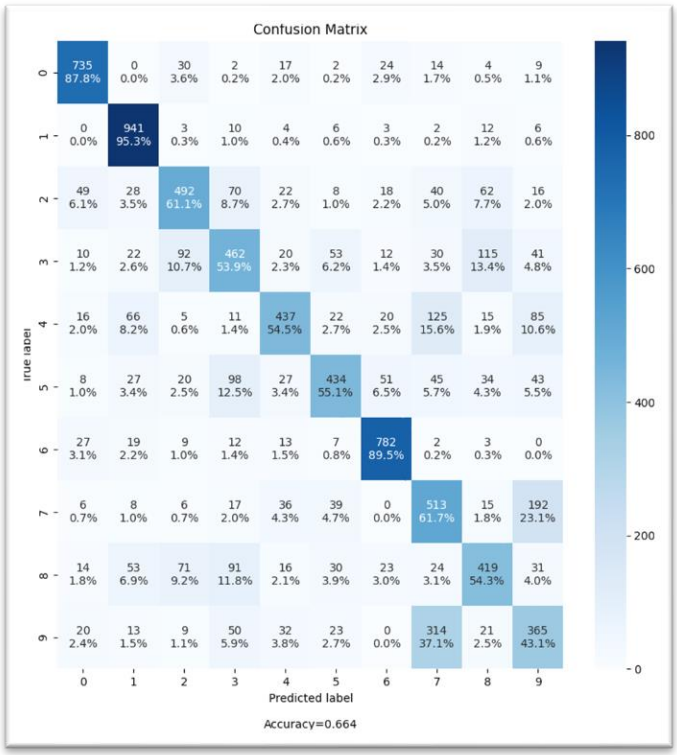
3. I ultimately decided to configure KNN with the above parameters with the highest rank_test_score. However, since n_neighbors are limited, I decided to check again by changing to only n_neighbors.

4. Based on this graph, n_neighbor = 19 showed the highest test accuracy, but to select the simple model that showed better performance, I decided to select 11 as the parameter of the final KNN model.



Confusion maxtrix (Left: Train set, Right: Test set)





2. Build a neural network model to predict handwriting. Discuss the accuracy and some of the challenges that you had.

Data preparation and normalization

1. The same image data was used, but normalization was performed using tf.keras.utils.normalize included in tensorflow.
2. By defining the input shape of the neural network model, the shape of X_train provides the dimensions of this data.
3. I determined the number of unique classes in the training label y_train and examined how many neurons should be set in the output layer.

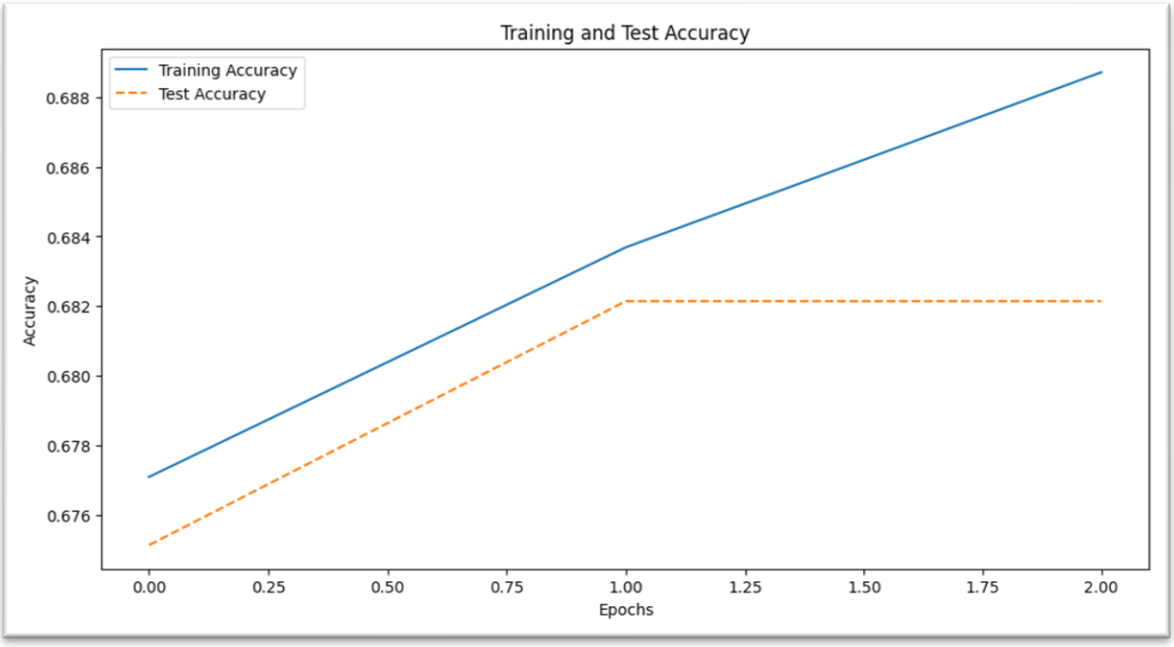
Setting number of layers and parameters

1. I used 'tf.keras.layers.Dense' following the video that implements a neural network in Python. Two dense layers with 128 neurons were added, and activation was set to ReLU.
2. A rectified linear unit (ReLU) is an activation function that introduces the property of nonlinearity to a deep learning model and solves the vanishing gradients issue. Here's why it's so popular (Krishnamurthy, 2022).
3. The final layer consisted of 10 neurons with numbers from 0 to 9. We configured probabilities for each class by configuring activation='softmax'.
4. The softmax activation function transforms the raw outputs of the neural network into a vector of probabilities, essentially a probability distribution over the input classes. Consider a multiclass classification problem with N classes (Priya, n.d.).

Training the model

1. I constructed the model using the 'model.complie' function.
2. Optimizer='adam': Adam is an optimization algorithm that can iteratively update network weights. The name is derived from adaptive moment estimation. The optimizer is called Adam because uses estimations of the first and second moments of the gradient to adapt the learning rate for each weight of the neural network (cornell, n.d.).
3. loss='sparse_categorical_crossentropy': This loss function is used when the target values are integers.
4. metrics=['accuracy']: Accuracy is used as a metric to monitor during training and testing.
5. I used training data X_train and trained the model using y_train labels for 3 generations.
6. After implementation, I used parameters to add verification data. The model is trained for another three epochs, but this time also using validation data X_test and y_test. The training history (loss, accuracy, validation loss, validation accuracy) for that epoch is stored in the history variable.

Using tf.keras.utils.normalize Train vs Test accuracy in Epoch=3 with two 128 neuron layers

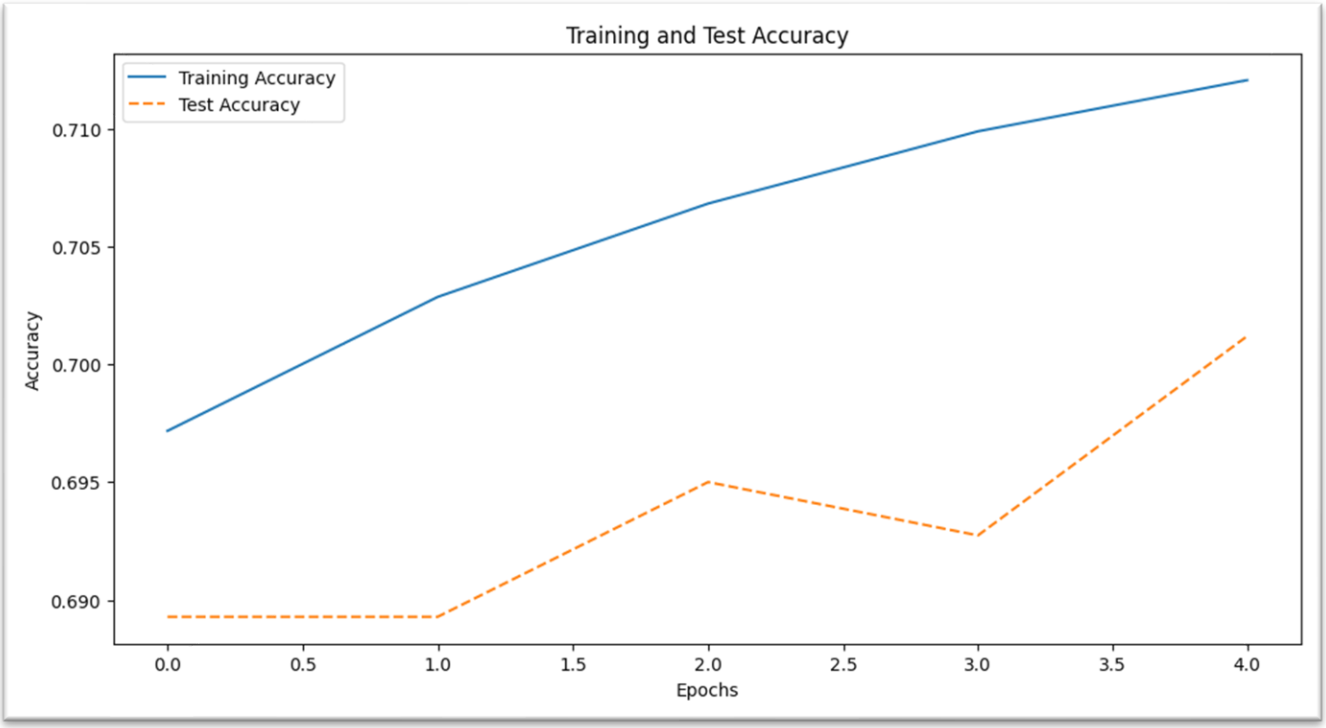


Looking at the results, you can see that as a result of learning a neural network with two layers with 128 neurons each, the train accuracy and test accuracy increase together until Epoch 1, but the gap between the two widens from Epoch 2. The final accuracy was found to be around 68.21%.

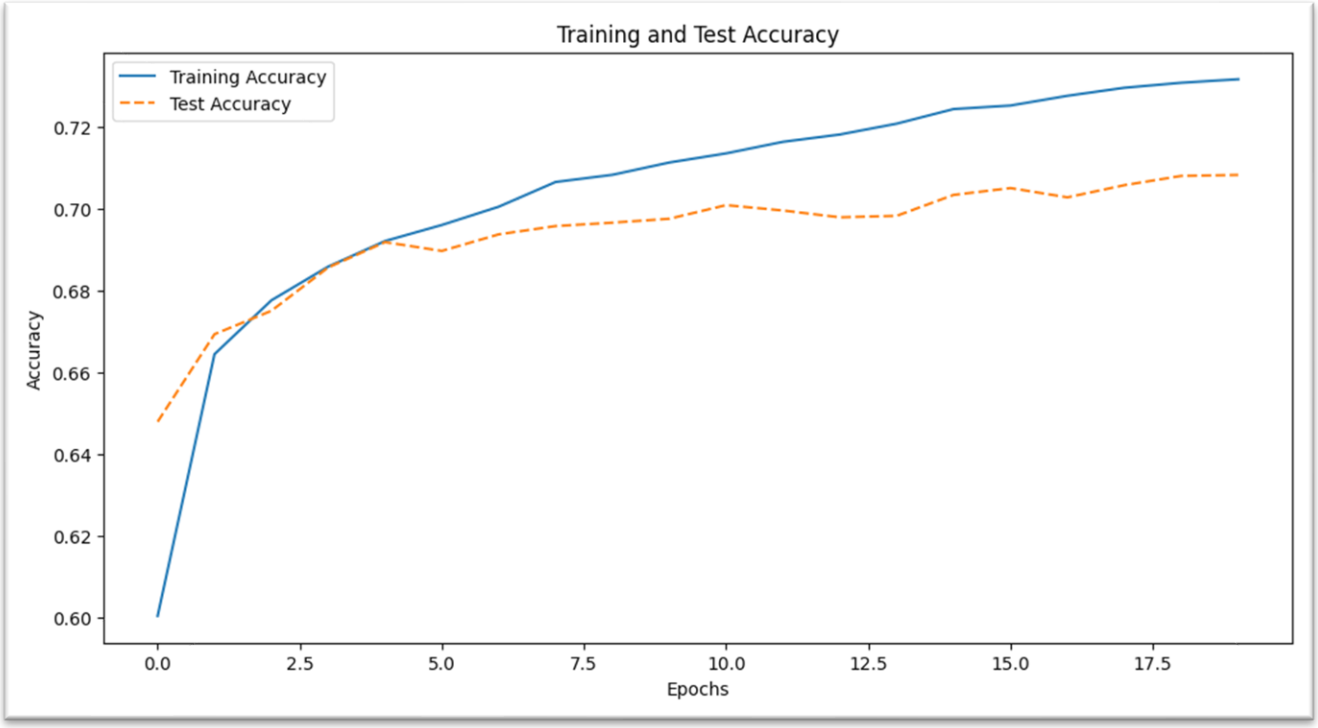
Increasing the Accuracy

1. To increase accuracy, I performed image-optimized normalization called rescaling.
2. I increased the epoch to 5 and achieved an accuracy of 70.11%.
3. I tried increasing the epoch to 20, but the accuracy did not improve significantly and recorded 70.83%.
4. Finally, increased the number of neurons included in each layer to 1024 and invested more time to construct a neural network, but the results were even worse, recording 62.79%.

Using Rescaling Train vs Test accuracy in Epoch=5 with two 128 neuron layers



Using Rescaling Train vs Test accuracy in Epoch=20 with two 128 neuron layers



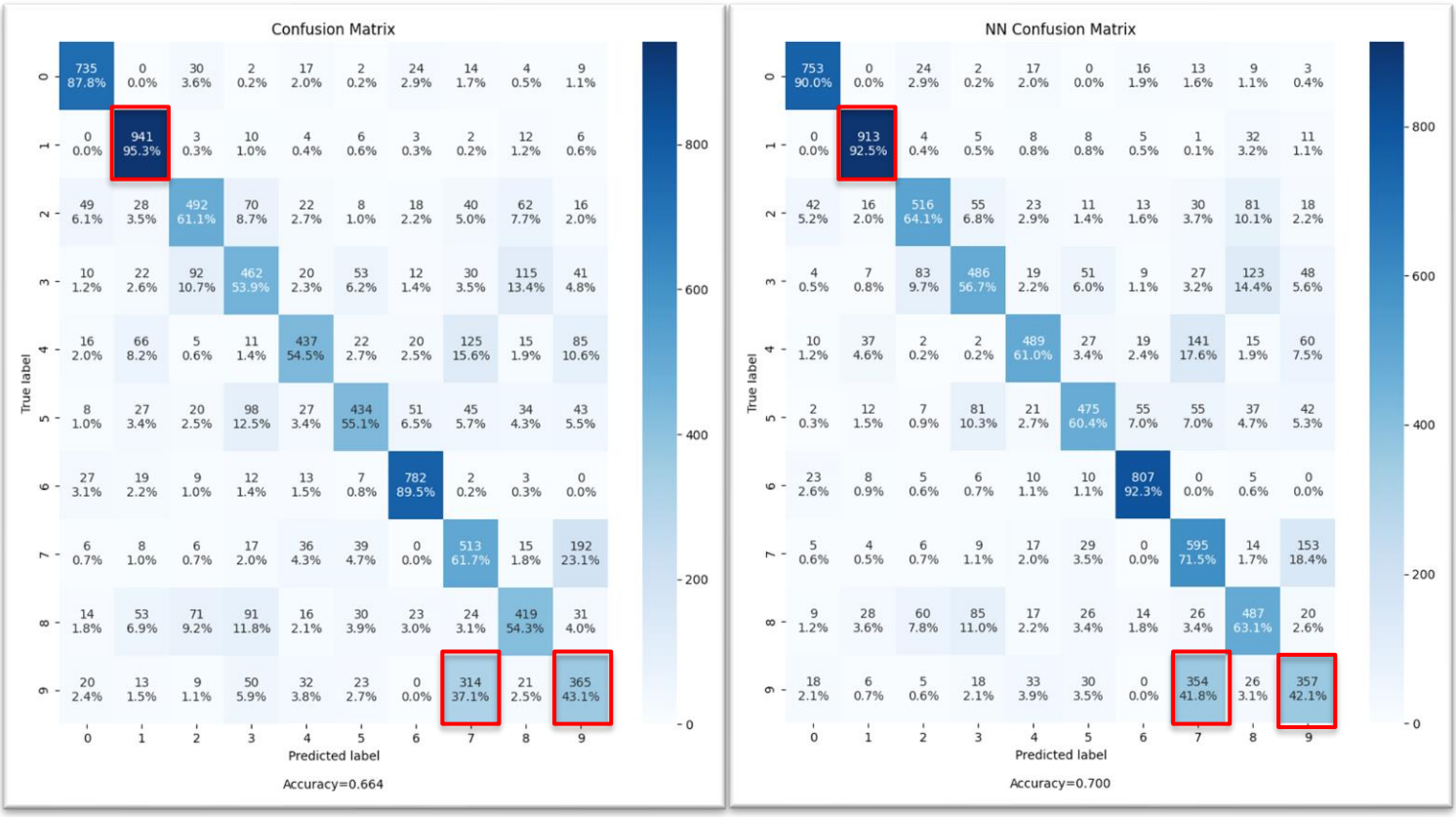
3. Using multiple benchmarking metrics to compare and contrast both models. Summarize your findings and suggest a final model for the school to use.

Accuracy

Metric	Value
Train Accuracy (KNN)	0.97747
Test Accuracy (KNN)	0.656905
Train Accuracy (NN)	0.713958
Test Accuracy (NN)	0.699762

If you look at the accuracy of the KNN model and the Neural Network (NN) model, you can see that the train accuracy is very high in KNN, but the test accuracy is low. Typically, overfitting appears. However, in the neural network, the train accuracy is not high, but it shows a more appropriately fitted model with almost similar test accuracy. Finally, the accruacy in the testset was 69.97% for the Neural network model, which is 4.28% higher than the 65.69% for the KNN model.

Confusion Matrix (Left: KNN, Right: Neural Network)



1. If you look at the confusion matrix in the testset, you can see that it shows similar tendencies for each number. While 0, 1, 6 are well distinguished, other numbers appear to be difficult to distinguish.
2. Looking at the individual numbers in detail, in a case like 8, KNN showed an accuracy of 54.3%, but NN showed an improved accuracy of 63.1%. This seems to be the reason why the NN model achieved an accuracy of 70%.
3. The best predicted number for each is 1, which is thought to be because it looks the most different from the other numbers. The most poorly predicted number was also 9 and was most often misinterpreted as 7.

Loss Function

I also considered a benchmark called the loss function, but since KNN has a structure that basically cannot have a loss function, it could not be compared with the loss function of the Neural Network.

KNN does not have a loss function that can be minimized during training. In fact, this algorithm is not trained at all. The only "training" that happens for KNN, is memorizing the data (creating a local copy), so that during prediction you can do a search and majority vote. Technically, no function is fitted to the data, and so, no optimization is done (it cannot be trained using gradient descent) (Tim, 2019).

Dataset Understanding

The data consists of a total of 46 columns and 42k rows. The target variable is the first column, label, which indicates where the handwriting number falls from 0 to 9. Pixels are distributed in a total of 45 columns. However, pixel information could not be imaged because accurate overall information was not provided, and it was difficult to determine the shape for imaging. Pixels were collected irregularly rather than continuously as follows. pixel43 pixel44 pixel92 pixel124 pixel125 pixel126 pixel127 pixel128 pixel129 pixel130 pixel131 pixel132 pixel133 pixel134 pixel135 pixel136 pixel137 pixel138 … pixel410 pixel411 pixel412 pixel413 pixel414 pixel415 pixel416 and pixel417. No missing data is confirmed.

Description of the variables/features in the dataset.

#	column name	Description
1	**label**	an integer from 0 to 9 and is the final result of the handwriting
2 - 46	**Pixels**	represent the grayscale of each pixel and has a number from 0 to 255

Headtail of Dataset

	label	pixel43	pixel44	pixel92	pixel124	...	pixel414	pixel415	pixel416	pixel417
0	1	0	0	0	0	...	0	0	0	0
1	0	0	0	0	137	...	254	17	0	0
2	1	0	0	0	3	...	0	0	0	0
3	4	0	0	0	0	...	0	0	0	0
4	0	0	0	0	155	...	253	253	129	0
41995	2	0	0	1	248	...	0	0	0	0
41996	0	0	0	0	0	...	0	0	0	0
41997	2	0	0	0	255	...	0	0	0	0
41998	2	0	0	0	255	...	0	0	0	0
41999	2	0	0	227	253	...	0	0	0	0

Exploratory Data Analysis

Descriptive Analysis of Dataset

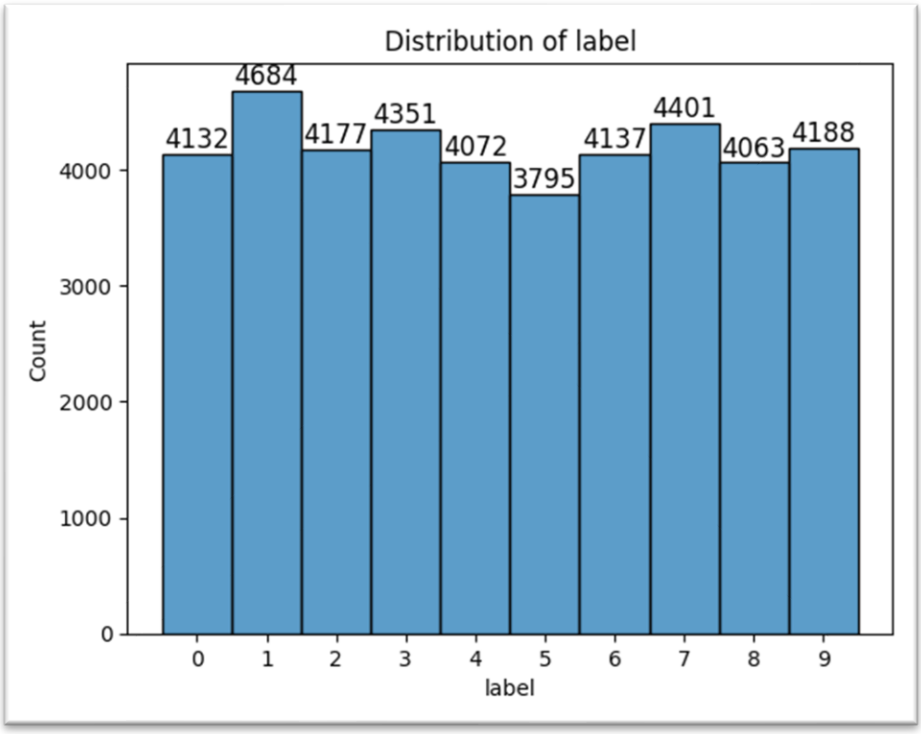
label	label	pixel43	pixel44	pixel92	pixel124	...	pixel414	pixel415	pixel416	pixel417
count	42000.0	42000.0	42000.0	42000.0	42000.0	...	42000.0	42000.0	42000.0	42000.0
mean	4.5	0.2	0.2	1.2	28.0	...	25.8	14.9	5.8	0.8
std	2.9	5.7	5.5	14.7	70.5	...	69.6	54.0	33.3	11.8
min	0	0	0	0	0	...	0	0	0	0
25%	2	0	0	0	0	...	0	0	0	0
50%	4	0	0	0	0	...	0	0	0	0
75%	7	0	0	0	0	...	0	0	0	0
max	9	255	255	255	255	...	255	255	255	255

1. The mean of the target variable label is 4.5, which is exactly the median between 0 and 9.
2. Each pixel has a value from 0 to 255. You can see that the very first pixels, 43 and 44, have a mean of 0 and are almost black.
3. We can assume that it is data located in a corner. Starting with pixel124, the mean may be over 20. You can see that the final pixel, 417, also has a low mean of 0.8 and is also composed of black.

Data Cleansing

1. I checked for missing values in the data and confirmed that there are no missing values.
2. When using Knn, I performed normalization using MinMaxScaler.
3. When doing neural network, I first used the 'tf.keras.utils.normalize' function included in tensorflow.
4. To make the results better, I was able to slightly increase the final accuracy by using rescaling appropriate for grayscale.

Data Visualizations



Looking at the label count, 1 is the most with 4,684. Considering that 1/10 of 42,000 is 4,200, this can be seen as a 10% higher figure. Since 1 appeared to be the easiest number to distinguish in the final results, it is believed that the number of labels may have affected accuracy. 7 was ultimately predicted to be confusingly similar to 9. Looking at this, it seems that the large number of 7s (4,401) also affected the accuracy.

Conclusion

In this project, I implemented the KNN model and compared it with the neural network (NN) model. By implementing the KNN model, I was able to learn about the characteristics of the KNN model once again and gain insight into the number of layers and neurons, which are characteristics of the neural network model. While constructing a neural network model, I searched for several articles on how to increase accuracy and tried to apply them. I learned that normalization is important in both KNN and NN models, and that accuracy slightly increased when rescaling was used.

Reference

NeuralNine. (2021). Neural network Python project - handwritten digit recognition. YouTube. Retrieved from https://www.youtube.com/watch?v=bte8Er0QhDg

Machine Learning Knowledge. (n.d.). KNN Classifier in Sklearn using GridSearchCV with example. Retrieved from https://machinelearningknowledge.ai/knn-classifier-in-sklearn-using-gridsearchcv-with-example/

Srinivasav, V. (2020). Digit recognition using KNN. Kaggle. Retrieved from https://www.kaggle.com/code/srinivasav22/digit-recognition-using-knn/notebook

Apple Inc. (n.d.). Improving your model's accuracy. Apple Developer Documentation. Retrieved from https://developer.apple.com/documentation/createml/improving-your-model-s-accuracy

Sharma, P. (2019, August 24). Why is scaling required in KNN and K-means? Medium. Retrieved from https://medium.com/analytics-vidhya/why-is-scaling-required-in-knn-and-k-means-8129e4d88ed7

Brownlee, J. (2020, August 28). How to use StandardScaler and MinMaxScaler transforms in Python. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/

ashwinsharmap. (2023, April 24). StandardScaler, MinMaxScaler and RobustScaler techniques | ML. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/

Loukas, S. (2020, May 28). Everything you need to know about Min-Max normalization in Python. Medium. Retrieved from https://towardsdatascience.com/everything-you-need-to-know-about-min-max-normalization-in-python-b79592732b79

TensorFlow. (n.d.). tf.keras.layers.Rescaling. Retrieved from https://www.tensorflow.org/api_docs/python/tf/keras/layers/Rescaling

WVJoe. (2021, February 2). model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test) isn't working. Stack Overflow. Retrieved from https://stackoverflow.com/questions/66016931/model-fitx-train-y-train-epochs-5-validation-data-x-test-y-test-isnt-wor

Hale, J. (2019). Scale, Standardize, or Normalize with Scikit-Learn. Medium. Retrieved from https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02#:~:text=let's%20start%20scaling!-,MinMaxScaler,shape%20of%20the%20original%20distribution.

Filho, D. (2023). Is feature scaling required for the KNN algorithm? Forecastegy. Retrieved from https://forecastegy.com/posts/is-feature-scaling-required-for-the-knn-algorithm/#:~:text=To%20put%20it%20simply%2C%20yes,based%20algorithms%20like%20the%20KNN.

Tracyrenee. (2023, March 6). The different normalisation functions in TensorFlow. Medium. Retrieved from https://medium.com/mlearning-ai/the-different-normalisation-functions-in-tensorflow-2e1f8a235f82#:~:text=pre%2Dtrained%20models.-,tf.,axis%20is%20equal%20to%201.

Patwardhan, S. (2022, June 23). Simple understanding and implementation of KNN algorithm. Analytics Vidhya. Retrieved from https://www.analyticsvidhya.com/blog/2021/04/simple-understanding-and-implementation-of-knn-algorithm/#:~:text=At%20low%20K%20values%2C%20there,high%20at%20lower%20K%20values.

Mosse, B. (2020, May 12). Why you may be getting low test accuracy: Try this quick way of comparing the distribution of the datasets. Medium. Retrieved from https://towardsdatascience.com/why-you-may-be-getting-low-test-accuracy-try-this-quick-way-of-comparing-the-distribution-of-the-9f06f5a72cfc

Krishnamurthy, A. (2022). The Rectified Linear Unit (ReLU) Activation Function. Built In. Retrieved from https://builtin.com/machine-learning/relu-activation-function

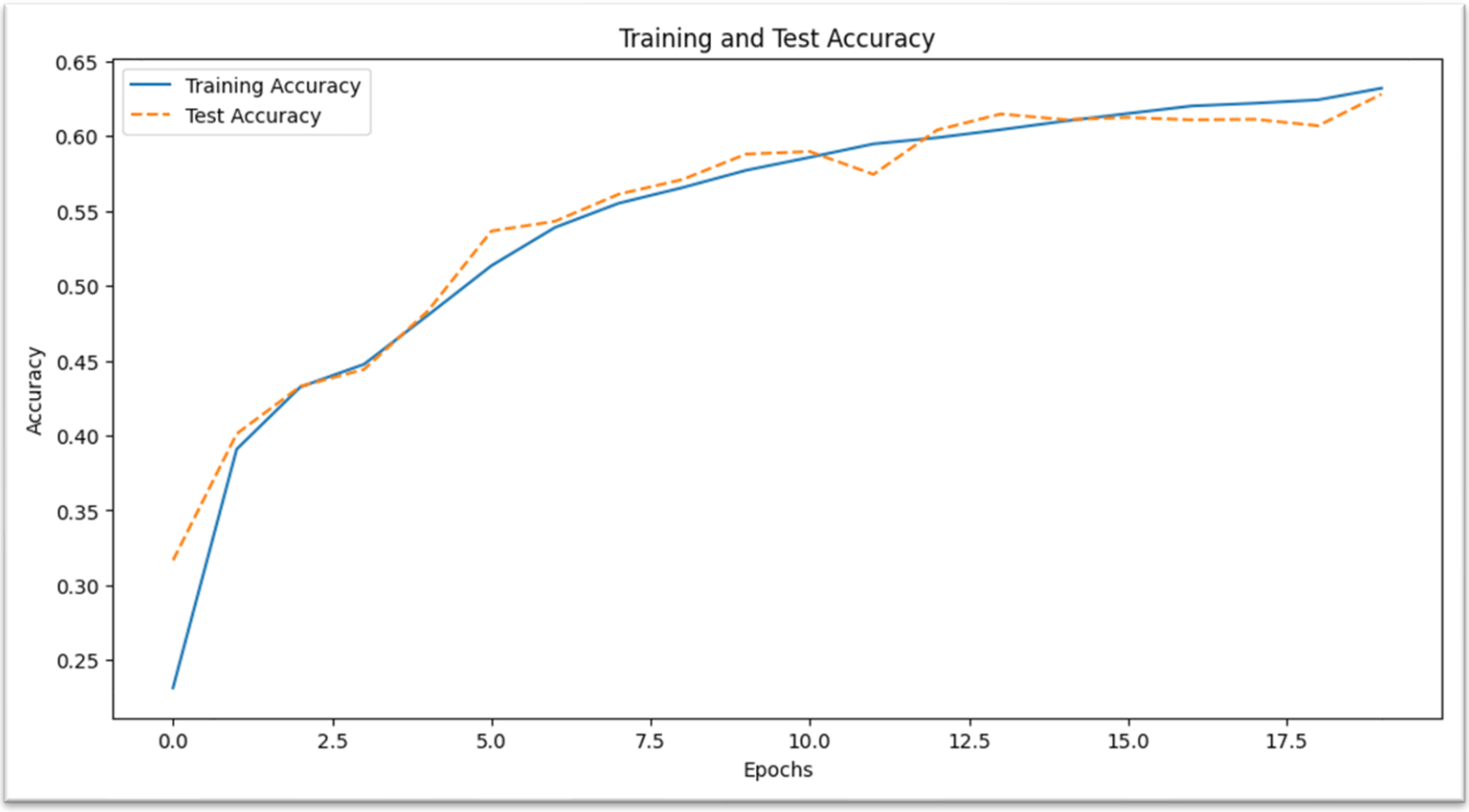
Priya, S. (n.d.). Softmax Activation Function: A Complete Guide. Pinecone. Retrieved from https://www.pinecone.io/learn/softmax-activation/#:~:text=The%20softmax%20activation%20function%20transforms,

Cornell. (n.d.). Adam, Optimization for Machine Learning. Retrieved from https://optimization.cbe.cornell.edu/index.php?title=Adam#:~:text=The%20name%20is%20derived%

Günay, G. (2020). Understanding parameters of KNN. Kaggle. Retrieved from https://www.kaggle.com/code/gorkemgunay/understanding-parameters-of-knn

Appendix (graphs):

Using Rescaling Train vs Test accuracy in Epoch=20 with two 1024 neuron layers



Appendix (Python code):

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from collections import Counter
import seaborn as sns

"""### Data Import"""

df = pd.read_csv('letters.csv')

df.head()

df.tail()

"""### Understanding Dataset"""

def missing_values(df):
    missing_number = df.isnull().sum().sort_values(ascending=False)
    missing_percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
    missing_values = pd.concat([missing_number, missing_percent], axis=1, keys=['Missing_Number',
'Missing_Percent'])
    return missing_values[missing_values['Missing_Number']>0]

def first_looking(df):
    print(colored("Shape:", attrs=['bold']), df.shape,'\n',
          colored('-'*79, 'red', attrs=['bold']),
          colored("\nInfo:\n", attrs=['bold']), sep='')
    print(df.info(), '\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("Number of Uniques:\n", attrs=['bold']), df.nunique(),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("Missing Values:\n", attrs=['bold']), missing_values(df),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("All Columns:", attrs=['bold']), list(df.columns),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')

    df.columns= df.columns.str.lower().str.replace('&', '_').str.replace(' ', '_')

    print(colored("Columns after rename:", attrs=['bold']), list(df.columns),'\n',
              colored('-'*79, 'red', attrs=['bold']), sep='')

pip install colorama

import colorama
from colorama import Fore, Style  # maakes strings colored
from termcolor import colored

missing_values(df)

first_looking(df)

df.describe()

"""#### Visualization"""

ax = sns.histplot(x=df['label'], discrete=True)
for p in ax.patches:
    ax.text(p.get_x() + p.get_width() / 2., p.get_height(), '%d' % int(p.get_height()),
            fontsize=12, ha='center', va='bottom')

plt.xticks(range(10))
plt.title("Distribution of label")
plt.show()

"""### Split the dataset into train and test"""
```

```
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler

# Divide X and y
X = df.drop('label', axis=1)
y = df['label']

# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train

"""### Try to Visualize but.."""

# Checking the number of first column
X_train.shape[1]

num_pixels=X_train.shape[0]

# Making the sqrt of pixels shape
side_length = int(np.sqrt(num_pixels))

# Checking the side_length and num_pixels
side_length, num_pixels

# Random selection
s = np.random.choice(range(X_train.shape[0]), size=12)

"""### KNN"""

from sklearn.neighbors import KNeighborsClassifier

# Normalization before

knn = KNeighborsClassifier(algorithm='brute', n_neighbors=5, p=2, weights='distance')

knn.fit(X_train, y_train)

"""#### confusion matrix"""

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

pred_train = knn.predict(X_train)
cm_test = confusion_matrix(y_true=y_train, y_pred=pred_train)
cm_test

train_accuracy = accuracy_score(y_train, pred_train)
train_accuracy

def make_confusion_matrix(cf,
                          group_names=None,
                          categories='auto',
                          count=True,
                          percent=True,
                          cbar=True,
                          xyticks=True,
                          xyplotlabels=True,
                          sum_stats=True,
                          figsize=None,
                          cmap='Blues',
                          title=None):
    '''
    This function will make a pretty plot of an sklearn Confusion Matrix cm using a Seaborn heatmap visualization.
```

```
    Arguments
    ---------
    cf:            confusion matrix to be passed in

    group_names:   List of strings that represent the labels row by row to be shown in each square.

    categories:    List of strings containing the categories to be displayed on the x,y axis. Default is 'auto'

    count:         If True, show the raw number in the confusion matrix. Default is True.

    normalize:     If True, show the proportions for each category. Default is True.

    cbar:          If True, show the color bar. The cbar values are based off the values in the confusion matrix.
                   Default is True.

    xyticks:       If True, show x and y ticks. Default is True.

    xyplotlabels:  If True, show 'True Label' and 'Predicted Label' on the figure. Default is True.

    sum_stats:     If True, display summary statistics below the figure. Default is True.

    figsize:       Tuple representing the figure size. Default will be the matplotlib rcParams value.

    cmap:          Colormap of the values displayed from matplotlib.pyplot.cm. Default is 'Blues'
                   See http://matplotlib.org/examples/color/colormaps_reference.html

    title:         Title for the heatmap. Default is None.

    '''


    # CODE TO GENERATE TEXT INSIDE EACH SQUARE
    blanks = ['' for i in range(cf.size)]

    if group_names and len(group_names)==cf.size:
        group_labels = ["{}\n".format(value) for value in group_names]
    else:
        group_labels = blanks

    if count:
        group_counts = ["{0:0.0f}\n".format(value) for value in cf.flatten()]
    else:
        group_counts = blanks

    if percent:
        row_sums = cf.sum(axis=1, keepdims=True)
        percentage_matrix = cf / row_sums
        group_percentages = ["{0:.1%}".format(value) for value in percentage_matrix.flatten()]
    else:
        group_percentages = blanks

    box_labels = [f"{v1}{v2}{v3}".strip() for v1, v2, v3 in zip(group_labels,group_counts,group_percentages)]
    box_labels = np.asarray(box_labels).reshape(cf.shape[0],cf.shape[1])


    # CODE TO GENERATE SUMMARY STATISTICS & TEXT FOR SUMMARY STATS
    if sum_stats:
        #Accuracy is sum of diagonal divided by total observations
        accuracy  = np.trace(cf) / float(np.sum(cf))

        #if it is a binary confusion matrix, show some more stats
        if len(cf)==2:
            #Metrics for Binary Confusion Matrices
            precision = cf[1,1] / sum(cf[:,1])
            recall    = cf[1,1] / sum(cf[1,:])
            f1_score  = 2*precision*recall / (precision + recall)
            stats_text = "\n\nAccuracy={:0.3f}\nPrecision={:0.3f}\nRecall={:0.3f}\nF1 Score={:0.3f}".format(
                accuracy,precision,recall,f1_score)
        else:
```

```
            stats_text = "\n\nAccuracy={:0.3f}".format(accuracy)
    else:
        stats_text = ""


    # SET FIGURE PARAMETERS ACCORDING TO OTHER ARGUMENTS
    if figsize==None:
        #Get default figure size if not set
        figsize = plt.rcParams.get('figure.figsize')

    if xyticks==False:
        #Do not show categories if xyticks is False
        categories=False


    # MAKE THE HEATMAP VISUALIZATION
    plt.figure(figsize=figsize)
    sns.heatmap(cf,annot=box_labels,fmt="",cmap=cmap,cbar=cbar,xticklabels=categories,yticklabels=categories)

    if xyplotlabels:
        plt.ylabel('True label')
        plt.xlabel('Predicted label' + stats_text)
    else:
        plt.xlabel(stats_text)

    if title:
        plt.title(title)

make_confusion_matrix(cm_test, categories=[str(i) for i in range(10)], cmap='Blues', title='Confusion Matrix',
figsize=(10, 10))

"""#### Apply to testset"""

pred_test = knn.predict(X_test)

test_accuracy = accuracy_score(y_test, pred_test)
test_accuracy

cm_test2 = confusion_matrix(y_true=y_test, y_pred=pred_test)
cm_test2

make_confusion_matrix(cm_test2, categories=[str(i) for i in range(10)], cmap='Blues', title='Confusion Matrix',
figsize=(10, 10))

"""#### Train and Test accuracy comparison"""

train_accuracies = []
test_accuracies = []

k_values = list(range(1, 31))

for k in k_values:
    knn = KNeighborsClassifier(algorithm='brute', n_neighbors=k, p=2, weights='distance')
    knn.fit(X_train, y_train)

    pred_train = knn.predict(X_train)
    train_accuracy = accuracy_score(y_train, pred_train)

    pred_test = knn.predict(X_test)
    test_accuracy = accuracy_score(y_test, pred_test)

    train_accuracies.append(train_accuracy)
    test_accuracies.append(test_accuracy)

plt.figure(figsize=(12, 6))
plt.plot(k_values, train_accuracies, label='Train Accuracy', marker='o')
plt.plot(k_values, test_accuracies, label='Test Accuracy', marker='x')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.title('Train vs. Test Accuracy by n_neighbors (k)')
```

```
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()

plt.show()

"""#### Find Proper parameter for KNN"""

from sklearn.model_selection import GridSearchCV

knn = KNeighborsClassifier(algorithm='brute', n_neighbors=5, p=2, weights='distance')

knn.fit(X_train, y_train)

make_confusion_matrix(cm_test, categories=[str(i) for i in range(10)], cmap='Blues', title='Confusion Matrix',
figsize=(10, 10))

"""#### Apply to testset"""

pred_test = knn.predict(X_test)

test_accuracy = accuracy_score(y_test, pred_test)
test_accuracy

cm_test2 = confusion_matrix(y_true=y_test, y_pred=pred_test)
cm_test2

make_confusion_matrix(cm_test2, categories=[str(i) for i in range(10)], cmap='Blues', title='Confusion Matrix',
figsize=(10, 10))

"""#### with other paramiters"""
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'algorithm': ['ball_tree', 'kd_tree', 'brute'],
    'p': [1, 2]
}

knn = KNeighborsClassifier()

grid_search = GridSearchCV(knn, param_grid, cv=3, verbose=1, n_jobs=-1)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_params

results = pd.DataFrame(grid_search.cv_results_)

results.head()

results.sort_values(by='mean_test_score', ascending=False)

filtered_results = results[(results['param_weights'] == best_params['weights']) &
                           (results['param_algorithm'] == best_params['algorithm']) &
                           (results['param_p'] == best_params['p'])]

filtered_results.head()

plt.plot(filtered_results['param_n_neighbors'], filtered_results['mean_test_score'])

plt.title('Accuracy vs. Number of Neighbors')
plt.xlabel('Number of Neighbors')
plt.ylabel('Mean Test Score (Accuracy)')
plt.legend()
plt.grid(True)
plt.show()

"""#### Find Proper parameter for KNN
```

```
"""

# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

knn = KNeighborsClassifier(algorithm='brute', p=2, weights='distance')

k_range = [k for k in range(1, 31) if k % 2 != 0]
param_grid = {
    'n_neighbors':k_range,
    'weights': ['distance'],
    'algorithm': ['brute'],
    'p': [2]
}

grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy', return_train_score=False, verbose=1)

grid_search = grid.fit(X_train, y_train)

best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_ * 100

best_params, best_accuracy

results = pd.DataFrame(grid_search.cv_results_)

results.head()

plt.plot(results['param_n_neighbors'], results['mean_test_score'])

plt.title('Accuracy vs. Number of Neighbors')
plt.xlabel('Number of Neighbors')
plt.ylabel('Mean Test Score (Accuracy)')
plt.legend()
plt.grid(True)
plt.show()

results.sort_values(by='mean_test_score', ascending=False)



"""#### Find Proper parameter for KNN"""
# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

knn = KNeighborsClassifier(algorithm='brute', n_neighbors=11, p=2, weights='distance')

knn.fit(X_train, y_train)

make_confusion_matrix(cm_test, categories=[str(i) for i in range(10)], cmap='Blues', title='Confusion Matrix',
figsize=(10, 10))

"""#### Apply to testset"""

pred_test = knn.predict(X_test)

test_accuracy = accuracy_score(y_test, pred_test)
test_accuracy

cm_test2 = confusion_matrix(y_true=y_test, y_pred=pred_test)
cm_test2
```

```
make_confusion_matrix(cm_test2, categories=[str(i) for i in range(10)], cmap='Blues', title='Confusion Matrix',
figsize=(10, 10))

"""#### with auto param_grid"""
# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto'],
    'p': [1, 2]
}

knn = KNeighborsClassifier()

grid_search = GridSearchCV(knn, param_grid, cv=3, verbose=1, n_jobs=-1)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_params

results = pd.DataFrame(grid_search.cv_results_)

results.head()

results.sort_values(by='mean_test_score', ascending=False)

filtered_results = results[(results['param_weights'] == best_params['weights']) &
                           (results['param_algorithm'] == best_params['algorithm']) &
                           (results['param_p'] == best_params['p'])]

filtered_results.head()

plt.plot(filtered_results['param_n_neighbors'], filtered_results['mean_test_score'])

plt.title('Accuracy vs. Number of Neighbors')
plt.xlabel('Number of Neighbors')
plt.ylabel('Mean Test Score (Accuracy)')
plt.legend()
plt.grid(True)
plt.show()

"""#### Final Method"""
# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

knn = KNeighborsClassifier(algorithm='brute', n_neighbors=9, p=2, weights='distance')

knn.fit(X_train, y_train)



make_confusion_matrix(cm_test, categories=[str(i) for i in range(10)], cmap='Blues', title='Confusion Matrix',
figsize=(10, 10))

"""#### Apply to testset"""

pred_test = knn.predict(X_test)
```

```
test_accuracy = accuracy_score(y_test, pred_test)
test_accuracy

cm_test2 = confusion_matrix(y_true=y_test, y_pred=pred_test)
cm_test2

make_confusion_matrix(cm_test2, categories=[str(i) for i in range(10)], cmap='Blues', title='Confusion Matrix',
figsize=(10, 10))

"""#### Finding n_neighbors"""
# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

knn = KNeighborsClassifier(algorithm='brute', p=2, weights='distance')

k_range = [k for k in range(1, 31) if k % 2 != 0]
param_grid = dict(n_neighbors=k_range)

grid = GridSearchCV(knn, param_grid, algorithm='brute', p=2, weights='distance', cv=10, scoring='accuracy',
return_train_score=False, verbose=1)

grid_search = grid.fit(X_train, y_train)

best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_ * 100

best_params, best_accuracy

results = pd.DataFrame(grid_search.cv_results_)

results.head()

plt.plot(results['param_n_neighbors'], results['mean_test_score'])

plt.title('Accuracy vs. Number of Neighbors')
plt.xlabel('Number of Neighbors')
plt.ylabel('Mean Test Score (Accuracy)')
plt.legend()
plt.grid(True)
plt.show()

"""### Neural Network with tensorflow"""
import tensorflow as tf

# normalization
X_train = tf.keras.utils.normalize(X_train, axis=1)
X_test = tf.keras.utils.normalize(X_test, axis=1)

input_shape = X_train.shape[1:]

num_classes = len(set(y_train))

# Modeling

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=input_shape))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# Set optimizer
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.fit(X_train, y_train, epochs=3)
```

```
history = model.fit(X_train, y_train, epochs=3, validation_data=(X_test, y_test))

model.save('handwritten.model')

model = tf.keras.models.load_model('handwritten.model')

loss, accuracy = model.evaluate(X_test, y_test)

print(loss)
print(accuracy)

plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy', linestyle='--')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Test Accuracy')
plt.legend()
plt.show()

"""#### Increase the accuracy"""
# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

input_shape = X_train.shape[1:]

num_classes = len(set(y_train))

# Modeling
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Rescaling(1./255, input_shape=input_shape))
model.add(tf.keras.layers.Flatten(input_shape=input_shape))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))

# Set optimizer
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=5)

history = model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))
loss, accuracy = model.evaluate(X_test, y_test)

print(loss)
print(accuracy)

history.history['accuracy']
history.history['val_accuracy']

plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy', linestyle='--')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Test Accuracy')
plt.legend()
plt.show()

"""#### only epochs added"""

# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

input_shape = X_train.shape[1:]

num_classes = len(set(y_train))

# Modeling
```

```
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Rescaling(1./255, input_shape=input_shape))
model.add(tf.keras.layers.Flatten(input_shape=input_shape))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))

# Set optimizer
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test))

loss, accuracy = model.evaluate(X_test, y_test)

print(loss)
print(accuracy)

plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy', linestyle='--')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Test Accuracy')
plt.legend()
plt.show()

""""#### increase depth and epochs a lot""
# Modeling

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Rescaling(1./255, input_shape=input_shape))
model.add(tf.keras.layers.Flatten(input_shape=input_shape))
model.add(tf.keras.layers.Dense(1024, activation='relu'))
model.add(tf.keras.layers.Dense(1024, activation='relu'))
model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))

# Set optimizer
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test))

loss, accuracy = model.evaluate(X_test, y_test)

print(loss)
print(accuracy)

plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy', linestyle='--')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Test Accuracy')
plt.legend()
plt.show()

""""#### Many layers""""
# normalization
X_train = tf.keras.utils.normalize(X_train, axis=1)
X_test = tf.keras.utils.normalize(X_test, axis=1)

input_shape = X_train.shape[1:]

num_classes = len(set(y_train))

# Modeling

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=input_shape))
```

```
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# Set optimizer
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))

model.save('handwritten.model')

model = tf.keras.models.load_model('handwritten.model')

loss, accuracy = model.evaluate(X_test, y_test)

print(loss)
print(accuracy)

plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy', linestyle='--')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Test Accuracy')
plt.legend()
plt.show()

"""### Comparing Benchmarks"""

nn_predictions = np.argmax(model.predict(X_test), axis=-1)

nn_cm = confusion_matrix(y_test, nn_predictions)
make_confusion_matrix(nn_cm, categories=[str(i) for i in range(10)], cmap='Blues', title='NN Confusion Matrix',
figsize=(10, 10))

print("KNN Train Accuracy:", train_accuracies[-1])
print("KNN Test Accuracy:", test_accuracies[-1])
print("Neural Network Test Loss:", loss)
print("Neural Network Test Accuracy:", accuracy)

metrics_df = pd.DataFrame({
    'Metric': ['Train Accuracy (KNN)', 'Test Accuracy (KNN)', 'Train Accuracy (NN)', 'Test Accuracy (NN)'],
    'Value': [train_accuracies[-1], test_accuracies[-1], history.history['accuracy'][-1],
history.history['val_accuracy'][-1]]
})

metrics_df

def knn_loss(y_true, y_pred):
    return 1 - accuracy_score(y_true, y_pred)

train_loss = knn_loss(y_train, pred_train)

test_loss = knn_loss(y_test, pred_test)

print("KNN Train Loss:", train_loss)
print("KNN Test Loss:", test_loss)
```