12/18/2019

Letter Recognition

Project Report

University of New Haven

Done By:

Shreya Gopal Sundari

00686740

M.S DATA SCIENCE – FALL19

Professor:

Dr. Keith Dillon

Index:

1. Statement of Objectives:

The Letter Recognition dataset consists of a 26 capital letters (classes) in the English alphabet based on 20 different fonts and its associated black and white rectangular pixel display features.

The objective is to build an appropriate *deep neural network* model to this multi class data that will learn to correctly classify the letter categories associated with its feature vectors extracted from raster scan images of the letters.

2. Dataset Details:

The dataset is borrowed from the *UCI Machine Learning Repository*.

The black & white image of each letter was converted into 16 primitive numerical attributes (features) which were then scaled to fit into a range of integer values from 0 through 15.

Number of Instances: 20000

Number of Class: 26 (Multi class)

Number of Attributes: 17 (Letter category and 16 numeric features)

The below table defines attributes in the dataset:

No.	Attribute Name	Description
1	Letter	Capital letter (26 values from A to Z)
2	x-box	Horizontal position of box (integer)
3	y-box	Vertical position of box (integer)
4	Width	Width of box (integer)
5	High	Height of box (integer)
6	Onpix	Total # on pixels (integer)
7	x-bar	Mean x of on pixels in box (integer)
8	y-bar	Mean y of on pixels in box (integer)
9	x2bar	Mean x variance (integer)
10	y2bar	Mean y variance (integer)

11	Xybar	Mean x y correlation (integer)
12	x2ybr	Mean of x * x * y (integer)
13	xy2br	Mean of x * y * y (integer)
14	x-ege	Mean edge count left to right (integer)
15	Xegvy	Correlation of x-ege with y (integer)
16	y-ege	Mean edge count bottom to top (integer)
17	Yegvx	correlation of y-ege with x (integer)

Table 1: Attributes in the Dataset and its description

3. Approach:

The following steps are implemented to determine the better neural network model in Keras for the Letter Recognition problem:

1. Data Loading from the CSV file.
2. Understanding & Visualizing the data.
3. Preparing the data for the model.
4. Hand coding various generators with Data Augmentation
5. Building basic model.
6. Building various models and tuning hyperparameters.

3.1. Loading Dataset:

Initially, all the basic necessary libraries like Pandas, Keras, pyplot etc, for building a Keras neural network are imported into Jupyter Notebook. If any other libraries are required in the future, they can be imported accordingly.

The 20,000 samples of data in the CSV file are loaded into Pandas dataframe using read_csv() function. This function returns the data in the CSV file as a two-dimensional data structure with labeled axes, called dataframe as shown below:

	letter	xbox	ybox	width	height	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybar	xy2bar	xedge	xedgey	yedge	yedgex
0	T	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
1	I	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
2	D	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
3	N	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
4	G	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10

Figure 1: Dataframe showing few samples of data from CSV file

3.2. Data Understanding & Visualizing:

After successful storage of data in the dataframe, one can view the data in the tabular format and can access each part of data easily. From the dataframe, it is understood that the dataset has a text data (capital letter) column, *'letter'* which needs to be converted into numerical data.

This conversion can be achieved in many ways, one of which is using *'ord()'* function. The following figure shows the code snippet for required conversion and the resulting dataframe.

```
#Converting the Alphabet into its corresponding number starting from 0

data.letter = [ ord(x) - 65 for x in data.letter ]
data.head()
```

	letter	xbox	ybox	width	height	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybar	xy2bar	xedge	xedgey	yedge	yedgex
0	19	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
1	8	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
2	3	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
3	13	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
4	6	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10

Figure 2: Converted Dataframe showing few samples of data from CSV file

In order to observe the relation between each attribute of the dataset, a correlation heatmap is generated and is shown below:

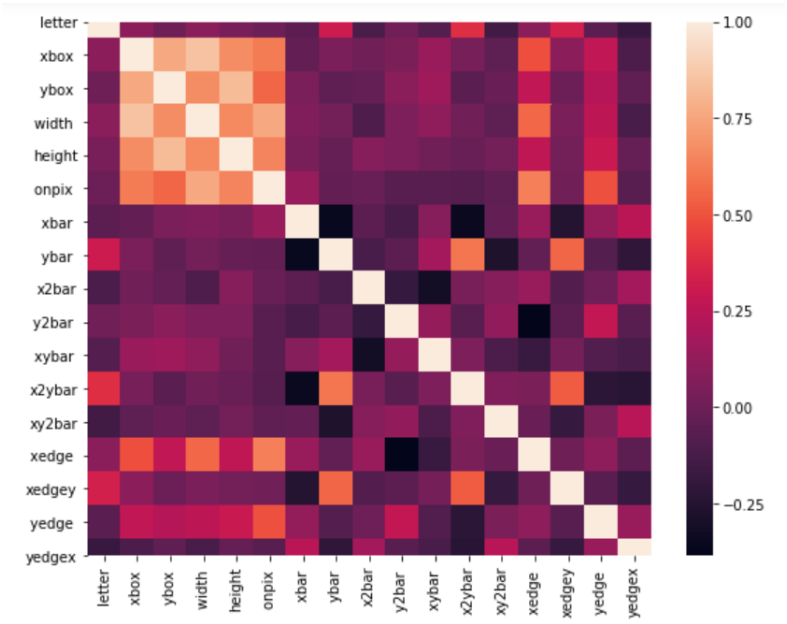


Figure 3: Distribution graphs of all attributes

To understand the distribution of all attributes in the given dataset, individual bar graphs are generated. The distribution graphs are shown below:

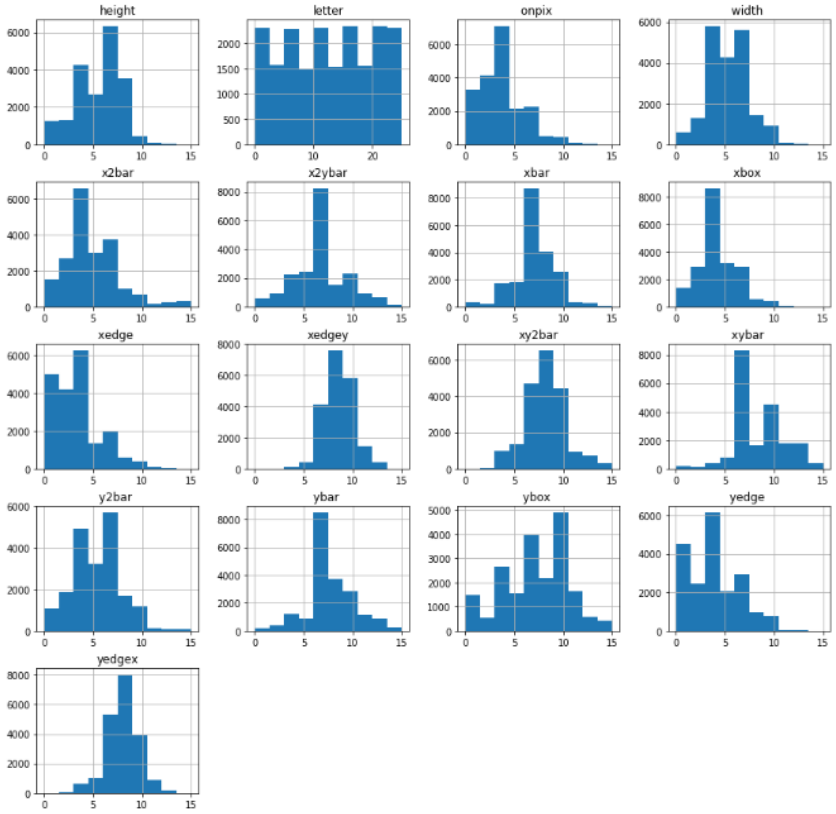


Figure 4: Distribution graphs of all attributes

From the above set of bar graphs, it is observed that all the attributes are in the range of 0 – 15 except the letter attribute. This letter attribute ranges from 0 to 25 as the data in it is capital alphabets and there are 26 of them.

3.3. Data Preparation:

3.3.1. Missing or Null Values:

In this step, we need to clean the dataset for it to be used in the model. So, initially each column of the dataset should be checked for any null or missing values by using the one of the two functions, isnull() and notnull(). Upon using these functions on the dataframe object, we got the values of 0 and 20,000 respectively for each attribute as shown below:

```
# Checking the data for null or missing values

data.isnull().sum()
```

```
letter    0
xbox      0
ybox      0
width     0
height    0
onpix     0
xbar      0
ybar      0
x2bar     0
y2bar     0
xybar     0
x2ybar    0
xy2bar    0
xedge     0
xedgey    0
yedge     0
yedgex    0
dtype: int64
```

```
# Checking the data for null or missing values

data.notnull().sum()
```

```
letter    20000
xbox      20000
ybox      20000
width     20000
height    20000
onpix     20000
xbar      20000
ybar      20000
x2bar     20000
y2bar     20000
xybar     20000
x2ybar    20000
xy2bar    20000
xedge     20000
xedgey    20000
yedge     20000
yedgex    20000
dtype: int64
```

Figure 5: Results of isnull() and notnull() functions

From the above results, it is clear that the Letter Recognition dataset used in this project doesn't have any null or missing values.

3.3.2. Creating X & y variables:

Now, split the dataframe into input (X) and target (y) variables. To form input variable, assign the dataframe to X and drop 'letter' column in it using dataframe *'drop()'* function. Target variable, y is formed by assigning the 'letter' column to it. Then these variables are passed into NN model.

3.3.3. Splitting Dataset to Train & Test Datasets:

After forming the input and target variables, the entire dataset needs to be split into train and test datasets. We train our model on the train data and test the accuracy of built model prediction or classification on the test data. By splitting the dataset, we are not doing any changes to the test data which gives unaltered results of our model efficiency.

The dataset splitting is done by using predefined *'train_test_split()'* function from scikit-learn as shown below. And the dataset is split into 70% of train data and 30% of test data.

The input and target variables of train data are denoted by X_train & y_train and for test data, X_test & y_test respectively.

```
# Spliting dataset into training (70%) and test (30%) set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
                                                    random_state = 0)
```

Figure 6: min-max & Z Normalization and their range

3.3.4. Feature Scaling or Normalizing:

The Letter Recognition dataset is a scaled set and is fit to a range of integer values from 0 to 15. But further feature scaling or normalization is performed on the dataset by using Z normalization and min-max normalization methods.

In Z normalization method, mean and standard deviation of each feature are used for scaling whereas in min-max normalization, minimum and maximum values of each feature are considered. Then the further calculation results scaled features.

On checking the scaled features by Z normalization, feature values ranged from -6 to 6 (approximately) and by min-max normalization, the range is 0 to 1. The scaled feature range is shown below:

```
# Normalizing train and test data separately
# min-max normalization
min_max_scaler = preprocessing.MinMaxScaler()
X_train_minmax = min_max_scaler.fit_transform(X_train)
X_test_minmax = min_max_scaler.fit_transform(X_test)
print("Train: min-max Normalization Range:", np.amin(X_train_minmax),"-", np.amax(X_train_minmax))
print("Test: min-max Normalization Range:", np.amin(X_test_minmax),"-", np.amax(X_test_minmax))
```

```
Train: min-max Normalization Range: 0.0 - 1.0
Test: min-max Normalization Range: 0.0 - 1.0
```

Figure 7: min-max Normalization and its range – Train & Test data

```
# Normalizing train and test data separately
# Z Normalization
sc = StandardScaler()
X_train_norm = sc.fit_transform(X_train)
X_test_norm = sc.fit_transform(X_test)
print("Train: Z Normalization Range:", np.amin(X_train_norm),"-", np.amax(X_train_norm))
print("Test: Z Normalization Range:", np.amin(X_test_norm),"-", np.amax(X_test_norm))
```

```
Train: Z Normalization Range: -5.404615288017298 - 5.739363196435594
Test: Z Normalization Range: -4.717728987393372 - 5.2779433735853925
```

Figure 8: Z Normalization and its range – Train & Test data

3.4. Hand coded Generators:

For the sake of this project, two generators are hand coded. Among them, one generator includes a data augmentation operation. The other generator doesn't include any extra operations instead it just splits the dataset into blocks by the specified size.

The data augmentation operation performed by the generator is adding a random floating value from -1.0 to 1.0 to each & every feature value i.e., adding or subtracting a small amount of noise to or from every feature value.

Both the generators didn't affect the values in the target variable. After coding the generators function, generator objects are creating by passing test & train data input variables separately. So, all together, for two generators & two different scaling methods, 8 generator objects are formed.

Note: The code for generators is mentioned in the Appendix part of this document.

3.5. Basic Neural Network (NN) Model:

In this project, Perceptron model is considered as the basic model and for this, Keras code is written without any hidden layers. The model is shown below:

```
# defining network model

model1 = Sequential() # model as a list (sequence) of layers
model1.add(layers.Dense(units = 26, activation = tf.nn.softmax, input_dim = 16))
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 26)                442
=================================================================
Total params: 442
Trainable params: 442
Non-trainable params: 0
_________________________________________________________________
```

Figure 9: NN model 1 – Perceptron

Model 1 architecture includes the following:

- Activation Function: Softmax
- Loss Function: sparse_categorical_crossentropy
- Optimiser: rmsprop
- Metrics: Accuracy
- Epochs: 10
- Batch_size: 200
- steps_per_epoch = 14 (fit_generator)
- steps = 6 (evaluate_generator)

Based on the accuracy obtained in this model using two different generators and normalization methods, further alterations are made to the architecture of the model to obtain better accuracy of prediction or classification. *Accuracy values are mentioned in the Table 2 in the Results section.*

3.6. Hyperparameter Tuning & Other NN Models:

By observing the initial model accuracy values, in the 2nd model, three hidden layers are added and few hyperparameter values are changed. Model 2 architecture includes the following:

- Hidden Layers: 3
- Hidden Activation: Relu
- Activation Function: Softmax
- Loss Function: sparse_categorical_crossentropy
- Optimiser: Adam
- Metrics: Accuracy
- Epochs: 20
- Batch_size: 100
- steps_per_epoch = 21 (fit_generator)
- steps = 18 (evaluate_generator)

Accuracy details for this model are mentioned in the Results section.

4. Results:

The following table shows the accuracy values for all the variations including generators for every NN model:

		No Generator		Generator 1		Generator 2	
		Z norm	**Min-max norm**	**Z norm**	**Min-max norm**	**Z norm**	**Min-max norm**
Model 1	**Trian**	0.6000	0.5488	0.6464	0.5996	0.4929	0.5066
	Test	0.6104	0.5472	0.6560	0.6072	0.4946	0.0872
Model 2	**Trian**	0.9219	0.8959	0.9318	0.9115	0.5369	0.5706
	Test	0.9035	0.8606	0.9060	0.8692	0.5400	0.0812

Table 2: Model Accuracy Values

Observations:

- In both the models with different generator options, Z normalized data option has high chances of classifying the data precisely.
- Upon adding noise to the data, the accuracy reduced compared to another instances.
- On increasing the hidden layers and tuning the hyperparameters accuracy of classification increased irrespective of the normalization method and generators used.
- On building few more models similarly, one can reach desired accuracy range.

5. References:

[1] P. W. Frey and D. J. Slate. "Letter Recognition Using Holland-style Adaptive Classifiers".

A Holland-style classifier system was applied to a complex letter recognition task using the above-mentioned dataset. The research focused on machine induction techniques for generating IF-THEN classifiers in which the IF part was a list of values for each of the 16 attributes and the THEN part was the correct category, i.e., one of the 26 letters of the alphabet.

[2] Deep Learning with Python Book by Francois Chollet

(https://www.manning.com/books/deep-learning- with-python)

[3] https://stats.stackexchange.com/questions/337237/order-of-normalization-augmentation-for-image-classification

[4] https://keras.io/models/sequential/

[5] https://medium.com/@anuj_shah/creating-custom-data-generator-for-training-deep-learning-models-part-1-5c62b20cff26

[6] https://medium.com/@anuj_shah/creating-custom-data-generator-for-training-deep-learning-models-part-2-be9ad08f3f0e

[7] https://medium.com/@anuj_shah/creating-custom-data-generator-for-training-deep-learning-models-part-3-c239297cd5d6

[8] https://stackoverflow.com/questions/28064634/random-state-pseudo-random-number-in-scikit-learn

[9] https://pandas-docs.github.io/pandas-docs-travis/user_guide/visualization.html

[10] https://www.geeksforgeeks.org/python-replace-negative-value-with-zero-in-numpy-array/

Appendix:

The source code for this project is mentioned below:

Letter Recognition:

- The Letter Recognition dataset consist of 20,000 data samples with 16 input values and one target value for each sample.
- The main objective of this project is to model a keras neural network to predict the letter/alphabet when 16 features are inputed.
- Initially, the dataset is analysed to understand it, and then the step by step process for building the neural network is applied.
- Here, variety of different architecture variations and various hyperparameters are also considered.

Importing Libraries:

In [2]:
```
# importing required libraries

import numpy as np
import pandas as pd
import random
import keras
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from keras.models import Sequential
from keras import layers
from keras import regularizers
```

Loding Dataset to Dataframe:

In [3]:
```
# loading the dataset from CSV file

data = pd.read_csv('letter-recognition.csv')
data.head()
```

Out[3]:

	letter	xbox	ybox	width	height	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybar	xy2bar	xedge	xedgey	yedge	y
0	T	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	
1	I	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	
2	D	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	
3	N	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	
4	G	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	

Data Understanding:

In [4]:
```
# finding the count of samples for each alphabet
# Please execute only once

data['letter'].value_counts()
```

Out[4]:
```
U    813
D    805
P    803
T    796
M    792
A    789
X    787
Y    786
Q    783
N    783
F    775
G    773
E    768
B    766
V    764
L    761
R    758
I    755
O    753
W    752
S    748
J    747
K    739
C    736
H    734
Z    734
Name: letter, dtype: int64
```

In [5]:
```
#Converting the Alphabet into its corresponding number starting from 0

data.letter = [ ord(x) - 65 for x in data.letter ]
data.head()
```

Out[5]:

	letter	xbox	ybox	width	height	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybar	xy2bar	xedge	xedgey	yedge	y
0	19	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	
1	8	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	
2	3	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	
3	13	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	
4	6	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	

In [6]:
```
data.describe()
```

Out[6]:

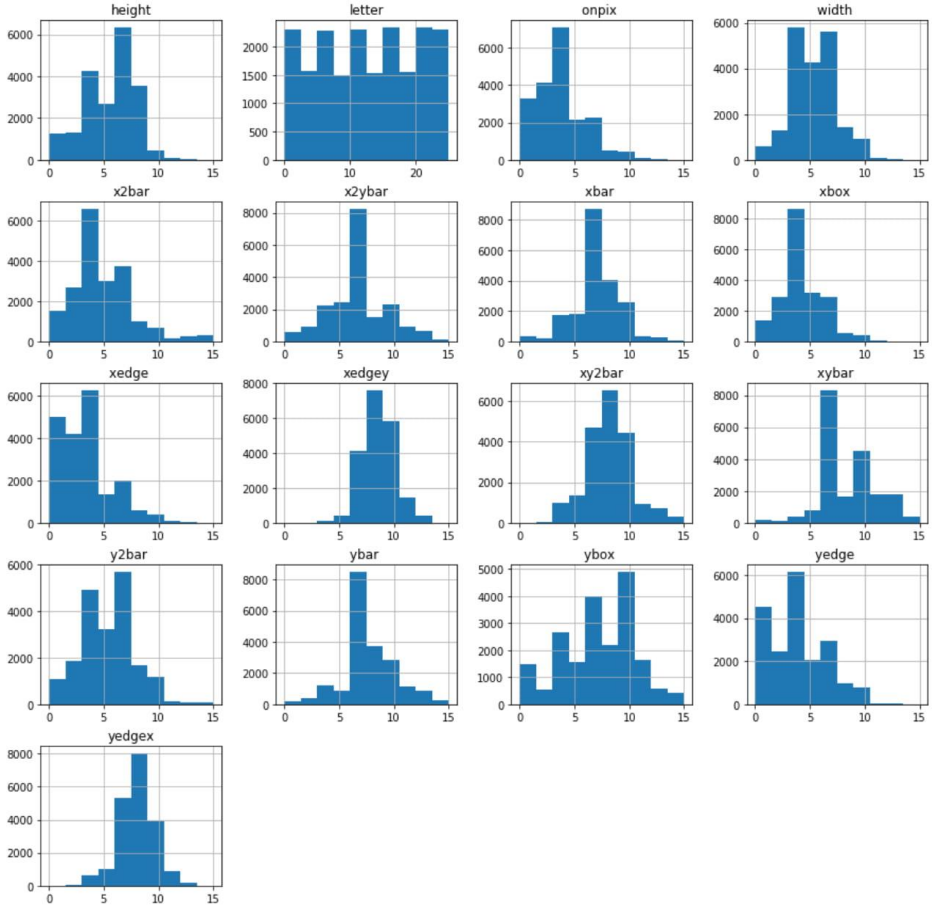
	letter	xbox	ybox	width	height	onpix	xbar	ybar
count	20000.000000	20000.000000	20000.000000	20000.000000	20000.00000	20000.000000	20000.000000	20000.000000
mean	12.516750	4.023550	7.035500	5.121850	5.37245	3.505850	6.897600	7.500450
std	7.502175	1.913212	3.304555	2.014573	2.26139	2.190458	2.026035	2.325354
min	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000
25%	6.000000	3.000000	5.000000	4.000000	4.00000	2.000000	6.000000	6.000000
50%	13.000000	4.000000	7.000000	5.000000	6.00000	3.000000	7.000000	7.000000
75%	19.000000	5.000000	9.000000	6.000000	7.00000	5.000000	8.000000	9.000000
max	25.000000	15.000000	15.000000	15.000000	15.00000	15.000000	15.000000	15.000000

The above results show that the letter recognition dataset doesn't have any null or missing values

Data Visualization:

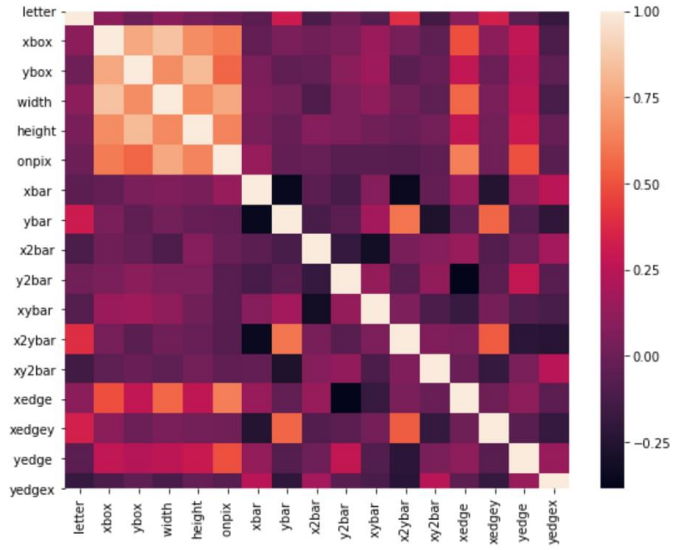
In [7]:
```
#Distribution bar graphs for each column
data.hist(figsize=(15,15))
```

Out[7]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001AA17EAD148>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA17EFC208>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA17F31B88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA181FBA08>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001AA18233A48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA1826BB48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA182A2C88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA182DAD88>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001AA182E6988>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA1831DB48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA18389088>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA183C0188>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001AA183F62C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA1842F3C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA18468488>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA1A76E5C8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001AA1A7A66C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA1A7DF7C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA1A8179C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AA1A84EB88>]],
      dtype=object)
```



In [8]:
```
# Heatmap of Correlation matrix of all features
plt.figure(figsize = (9,7))
sns.heatmap(data.corr())
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1aa17ea8508>



Data Preparation & Normalization:

In [9]:
```
# Checking the data for null or missing values

data.isnull().sum()
```

Out[9]:
```
letter    0
xbox      0
ybox      0
width     0
height    0
onpix     0
xbar      0
ybar      0
x2bar     0
y2bar     0
xybar     0
x2ybar    0
xy2bar    0
xedge     0
xedgey    0
yedge     0
yedgex    0
dtype: int64
```

In [10]:
```
# Checking the data for null or missing values

data.notnull().sum()
```

Out[10]:
```
letter    20000
xbox      20000
ybox      20000
width     20000
height    20000
onpix     20000
xbar      20000
ybar      20000
x2bar     20000
y2bar     20000
xybar     20000
x2ybar    20000
xy2bar    20000
xedge     20000
xedgey    20000
yedge     20000
yedgex    20000
dtype: int64
```

In [11]:
```
# split into input (X) and output (y) variables

X = data.drop('letter',axis=1)
y = data['letter']
X.head()
```

Out[11]:

	xbox	ybox	width	height	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybar	xy2bar	xedge	xedgey	yedge	yedgex
0	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
1	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
2	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
3	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
4	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10

In [12]:
```
y.head()
```

Out[12]:
```
0    19
1     8
2     3
3    13
4     6
Name: letter, dtype: int64
```

In [13]:
```
# Spliting dataset into training (70%) and test (30%) set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
                                                    random_state = 0)
```

In [14]:
```
# Normalizing train and test data separately
# min-max normalization
min_max_scaler = preprocessing.MinMaxScaler()
X_train_minmax = min_max_scaler.fit_transform(X_train)
X_test_minmax = min_max_scaler.fit_transform(X_test)
print("Train: min-max Normalization Range:", np.amin(X_train_minmax),"-", np.amax(X_train_minmax))
print("Test: min-max Normalization Range:", np.amin(X_test_minmax),"-", np.amax(X_test_minmax))
```

```
Train: min-max Normalization Range: 0.0 - 1.0
Test: min-max Normalization Range: 0.0 - 1.0
```

In [15]:
```
# Normalizing train and test data separately
# Z Normalization
sc = StandardScaler()
X_train_norm = sc.fit_transform(X_train)
X_test_norm = sc.fit_transform(X_test)
print("Train: Z Normalization Range:", np.amin(X_train_norm),"-", np.amax(X_train_norm))
print("Test: Z Normalization Range:", np.amin(X_test_norm),"-", np.amax(X_test_norm))
```

```
Train: Z Normalization Range: -5.404615288017298 - 5.739363196435594
Test: Z Normalization Range: -4.717728987393372 - 5.2779433735853925
```

Generators without and with Data Augmentation: Total 2 in number

In [16]:
```
#Creating generator to break the dataset into blocks
#Not including Data Augmentation
def myGenerator(features, label, batch_size):
    '''This generator splits the dataset into blocks
    of specified batch size. Without Data Augmentation'''

    num_samples = len(label)
    #features = features.to_numpy()
    label = label.to_numpy()
    #print(features, label)
    while True: # Loop forever so the generator never terminates

        for offset in range(0, num_samples, batch_size):
            #print("Range", offset, "-" , offset+batch_size)
            # Gets the samples you'll use in this batch
            X_batch = features[offset:offset+batch_size]
            y_batch = label[offset:offset+batch_size]

            # The generator-y part: yield the next training batch
            yield X_batch, y_batch
```

In [17]:
```
#Creating custom generator with data augmentation

def myGenAugmentNoise(features, label, batch_size):

    '''This generator adds a random floating value from -1 to 1 to
    each & every feature value. The label values are not altered.'''

    num_samples = len(label)
    #features = features.to_numpy()
    label = label.to_numpy()

    while True: # Loop forever so the generator never terminates

        for offset in range(0, num_samples, batch_size):
            #print("Range", offset, "-" , offset+batch_size)
            # Get the samples you'll use in this batch
            X_batch = features[offset:offset+batch_size]
            y_batch = label[offset:offset+batch_size]

            # Initialise X_train and y_train arrays for this batch
            X_train = []
            y_train = []

            # For each example
            for i in range(offset, offset+batch_size):
                row = []
                for j in range(len(features[0])):
                    row.append(features[i][j] + random.uniform(-1.0, 1.0))
                X_train.append(row)
                y_train.append(label[i])

            # Make sure they're numpy arrays (as opposed to lists)
            X_train = np.array(X_train)
            y_train = np.array(y_train)

            # The generator-y part: yield the next training batch
            yield X_train, y_train
```

In [18]:
```
#Generator objects for train data
#Z normalized data
train_gen11 = myGenerator(X_train_norm, y_train, 1000)
train_gen21 = myGenAugmentNoise(X_train_norm, y_train, 1000)

#min-max normalized data
train_gen12 = myGenerator(X_train_minmax, y_train, 1000)
train_gen22 = myGenAugmentNoise(X_train_minmax, y_train, 1000)
```

In [19]:
```
#Generator objects for test data
#Z normalized data
test_gen11 = myGenerator(X_test_norm, y_test, 1000)
test_gen21 = myGenAugmentNoise(X_test_norm, y_test, 1000)

#min-max normalized data
test_gen12 = myGenerator(X_test_minmax, y_test, 1000)
test_gen22 = myGenAugmentNoise(X_test_minmax, y_test, 1000)
```

Keras Model Building:

Variation 1:

- Architecture: **Perceptron**
- Activation Function: **Softmax**
- Loss Function: **sparse_categorical_crossentropy**
- Optimiser: **rmsprop**

In [20]:
```
# defining network model

model1 = Sequential() # model as a list (sequence) of layers
model1.add(layers.Dense(units = 26, activation = tf.nn.softmax, input_dim = 16))
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 26)                442
=================================================================
Total params: 442
Trainable params: 442
Non-trainable params: 0
_________________________________________________________________
```

In [21]:
```
model1.compile(loss='sparse_categorical_crossentropy', optimizer='rmsprop',
               metrics=['accuracy'])
```

Model 1 witout using generator:

In [22]:
```
# Training the model
#Witout generator # Z normalization

model1.fit(X_train_norm, y_train, batch_size = 200, epochs = 10)
```

```
WARNING:tensorflow:From C:\Users\sunda\Anaconda3\lib\site-packages\keras\backend\tensorflow_backe
nd.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables i
nstead.

Epoch 1/10
14000/14000 [==============================] - 0s 18us/step - loss: 3.3535 - accuracy: 0.0914
Epoch 2/10
14000/14000 [==============================] - 0s 12us/step - loss: 2.9807 - accuracy: 0.1772
Epoch 3/10
14000/14000 [==============================] - 0s 12us/step - loss: 2.6894 - accuracy: 0.2626
Epoch 4/10
14000/14000 [==============================] - 0s 12us/step - loss: 2.4556 - accuracy: 0.3439
Epoch 5/10
14000/14000 [==============================] - 0s 12us/step - loss: 2.2648 - accuracy: 0.4139
Epoch 6/10
14000/14000 [==============================] - 0s 12us/step - loss: 2.1074 - accuracy: 0.4686
Epoch 7/10
14000/14000 [==============================] - 0s 12us/step - loss: 1.9765 - accuracy: 0.5144
Epoch 8/10
14000/14000 [==============================] - 0s 13us/step - loss: 1.8663 - accuracy: 0.5497
Epoch 9/10
14000/14000 [==============================] - 0s 14us/step - loss: 1.7729 - accuracy: 0.5752
Epoch 10/10
14000/14000 [==============================] - 0s 12us/step - loss: 1.6943 - accuracy: 0.6000
```

Out[22]:
```
<keras.callbacks.callbacks.History at 0x1aa1ad7c888>
```

In [23]:
```
test_loss, test_acc = model1.evaluate(X_test_norm, y_test)

print('Test Data accuracy:', test_acc)
```

```
6000/6000 [==============================] - 0s 32us/step
Test Data accuracy: 0.6104999780654907
```

In [24]:
```
# Training the model
#Witout generator #min-max Normalization

model1.fit(X_train_minmax, y_train, batch_size = 200, epochs = 10)
```

```
Epoch 1/10
14000/14000 [==============================] - 0s 9us/step - loss: 2.9583 - accuracy: 0.1129
Epoch 2/10
14000/14000 [==============================] - 0s 10us/step - loss: 2.8907 - accuracy: 0.3577
Epoch 3/10
14000/14000 [==============================] - 0s 12us/step - loss: 2.8472 - accuracy: 0.4904
Epoch 4/10
14000/14000 [==============================] - 0s 13us/step - loss: 2.8134 - accuracy: 0.5106
Epoch 5/10
14000/14000 [==============================] - 0s 13us/step - loss: 2.7834 - accuracy: 0.5201
Epoch 6/10
14000/14000 [==============================] - 0s 13us/step - loss: 2.7553 - accuracy: 0.5457
Epoch 7/10
14000/14000 [==============================] - 0s 13us/step - loss: 2.7283 - accuracy: 0.5354
Epoch 8/10
14000/14000 [==============================] - 0s 13us/step - loss: 2.7024 - accuracy: 0.5446
Epoch 9/10
14000/14000 [==============================] - 0s 16us/step - loss: 2.6768 - accuracy: 0.5472
Epoch 10/10
14000/14000 [==============================] - 0s 15us/step - loss: 2.6518 - accuracy: 0.5488
```

Out[24]:
```
<keras.callbacks.callbacks.History at 0x1aa1af24d08>
```

In [25]:
```
test_loss, test_acc = model1.evaluate(X_test_minmax, y_test)

print('Test Data accuracy:', test_acc)
```

```
6000/6000 [==============================] - 0s 28us/step
Test Data accuracy: 0.547166645526886
```

Model 1 with generator 1:

In [26]:
```
#Training the model
#With generator 1 # Z normalization

model1.fit_generator(train_gen11, steps_per_epoch = 14, epochs = 10)
```

```
Epoch 1/10
14/14 [==============================] - 0s 4ms/step - loss: 1.5467 - accuracy: 0.5498
Epoch 2/10
14/14 [==============================] - 0s 4ms/step - loss: 1.5071 - accuracy: 0.5636
Epoch 3/10
14/14 [==============================] - 0s 4ms/step - loss: 1.4753 - accuracy: 0.5774
Epoch 4/10
14/14 [==============================] - 0s 4ms/step - loss: 1.4473 - accuracy: 0.5869
Epoch 5/10
14/14 [==============================] - 0s 4ms/step - loss: 1.4221 - accuracy: 0.5988
Epoch 6/10
14/14 [==============================] - 0s 4ms/step - loss: 1.3996 - accuracy: 0.6084
Epoch 7/10
14/14 [==============================] - 0s 4ms/step - loss: 1.3793 - accuracy: 0.6186
Epoch 8/10
14/14 [==============================] - 0s 4ms/step - loss: 1.3612 - accuracy: 0.6296
Epoch 9/10
14/14 [==============================] - 0s 4ms/step - loss: 1.3449 - accuracy: 0.6386
Epoch 10/10
14/14 [==============================] - 0s 4ms/step - loss: 1.3302 - accuracy: 0.6464
```

Out[26]:
```
<keras.callbacks.callbacks.History at 0x1aa1af24808>
```

In [27]:
```
test_loss, test_acc = model1.evaluate_generator(test_gen11, steps = 6)

print('Test Data accuracy:', test_acc)
```

```
Test Data accuracy: 0.656000018119812
```

In [28]:
```
#Training the model
#With generator 1 # min-max normalization

model1.fit_generator(train_gen12, steps_per_epoch = 14, epochs = 10)
```

```
Epoch 1/10
14/14 [==============================] - 0s 3ms/step - loss: 2.6757 - accuracy: 0.2792
Epoch 2/10
14/14 [==============================] - 0s 4ms/step - loss: 2.6560 - accuracy: 0.3479
Epoch 3/10
14/14 [==============================] - 0s 6ms/step - loss: 2.6410 - accuracy: 0.4182
Epoch 4/10
14/14 [==============================] - 0s 4ms/step - loss: 2.6285 - accuracy: 0.4821
Epoch 5/10
14/14 [==============================] - 0s 3ms/step - loss: 2.6177 - accuracy: 0.5294
Epoch 6/10
14/14 [==============================] - 0s 4ms/step - loss: 2.6080 - accuracy: 0.5641
Epoch 7/10
14/14 [==============================] - 0s 4ms/step - loss: 2.5993 - accuracy: 0.5836
Epoch 8/10
14/14 [==============================] - 0s 3ms/step - loss: 2.5911 - accuracy: 0.5906
Epoch 9/10
14/14 [==============================] - 0s 4ms/step - loss: 2.5833 - accuracy: 0.5974
Epoch 10/10
14/14 [==============================] - 0s 4ms/step - loss: 2.5759 - accuracy: 0.5996
```

Out[28]:
```
<keras.callbacks.callbacks.History at 0x1aa1ade7688>
```

In [29]:
```
test_loss, test_acc = model1.evaluate_generator(test_gen12, steps = 6)

print('Test Data accuracy:', test_acc)
```

```
Test Data accuracy: 0.6071666479110718
```

Model 1 with generator 2:

In [30]:
```
#Training the model
#With generator 2 # Z normalization

model1.fit_generator(train_gen21, steps_per_epoch = 14, epochs = 10)
```

```
Epoch 1/10
14/14 [==============================] - 1s 43ms/step - loss: 1.8582 - accuracy: 0.4654
Epoch 2/10
14/14 [==============================] - 1s 45ms/step - loss: 1.8264 - accuracy: 0.4720
Epoch 3/10
14/14 [==============================] - 1s 49ms/step - loss: 1.8165 - accuracy: 0.4723
Epoch 4/10
14/14 [==============================] - 1s 43ms/step - loss: 1.7903 - accuracy: 0.4774
Epoch 5/10
14/14 [==============================] - 1s 51ms/step - loss: 1.7730 - accuracy: 0.4791
Epoch 6/10
14/14 [==============================] - 1s 46ms/step - loss: 1.7511 - accuracy: 0.4901 0s - los
s: 1.7466 - accuracy: 0.49
Epoch 7/10
14/14 [==============================] - 0s 35ms/step - loss: 1.7439 - accuracy: 0.4844
Epoch 8/10
14/14 [==============================] - 1s 52ms/step - loss: 1.7307 - accuracy: 0.4863 0s - los
s: 1.7326 - accu
Epoch 9/10
14/14 [==============================] - 1s 39ms/step - loss: 1.7209 - accuracy: 0.4892
Epoch 10/10
14/14 [==============================] - 1s 47ms/step - loss: 1.7180 - accuracy: 0.4929
```

Out[30]:
```
<keras.callbacks.callbacks.History at 0x1aa1ade7108>
```

In [31]:
```
test_loss, test_acc = model1.evaluate_generator(test_gen21, steps = 6)

print('Test Data accuracy:', test_acc)
```

```
Test Data accuracy: 0.4945000112056732
```

In [32]:
```
#Training the model
#With generator 2 # min-max normalization

model1.fit_generator(train_gen21, steps_per_epoch = 14, epochs = 10)
```

```
Epoch 1/10
14/14 [==============================] - 1s 48ms/step - loss: 1.7167 - accuracy: 0.4939
Epoch 2/10
14/14 [==============================] - 0s 33ms/step - loss: 1.6978 - accuracy: 0.5009
Epoch 3/10
14/14 [==============================] - 1s 44ms/step - loss: 1.6973 - accuracy: 0.4988
Epoch 4/10
14/14 [==============================] - 1s 44ms/step - loss: 1.6873 - accuracy: 0.4991 0s - los
s: 1.7004 - accu
Epoch 5/10
14/14 [==============================] - 1s 52ms/step - loss: 1.6763 - accuracy: 0.4951
Epoch 6/10
14/14 [==============================] - 1s 47ms/step - loss: 1.6833 - accuracy: 0.4972
Epoch 7/10
14/14 [==============================] - 1s 47ms/step - loss: 1.6705 - accuracy: 0.5021
Epoch 8/10
14/14 [==============================] - 1s 45ms/step - loss: 1.6696 - accuracy: 0.5031
Epoch 9/10
14/14 [==============================] - 0s 34ms/step - loss: 1.6624 - accuracy: 0.5066 0s - los
s: 1.6657 - accuracy
Epoch 10/10
14/14 [==============================] - 1s 47ms/step - loss: 1.6528 - accuracy: 0.5066 0s - los
s: 1.6498 - accuracy:  - ETA: 0s - loss: 1.6509 - accuracy: 0.50
```

Out[32]:
```
<keras.callbacks.callbacks.History at 0x1aa1cea5a88>
```

In [33]:
```
test_loss, test_acc = model1.evaluate_generator(test_gen22, steps = 6)

print('Test Data accuracy:', test_acc)
```

```
Test Data accuracy: 0.0871666669845581
```

Variation 2:

- Architecture: **3 Hidden layers**
- Hidden Activation Function: **Relu**
- Activation Function: **Softmax**
- Loss Function: **sparse_categorical_crossentropy**
- Optimiser: **adam**

In [34]:
```
# defining network model

model3 = Sequential() # model as a list (sequence) of layers
model3.add(layers.Dense(units = 54, activation = tf.nn.relu, input_dim = 16))
model3.add(layers.Dense(units = 48, activation = tf.nn.relu))
model3.add(layers.Dense(units = 32, activation = tf.nn.relu))
model3.add(layers.Dense(units = 26, activation = tf.nn.softmax))
model3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 54)                918
_________________________________________________________________
dense_3 (Dense)              (None, 48)                2640
_________________________________________________________________
dense_4 (Dense)              (None, 32)                1568
_________________________________________________________________
dense_5 (Dense)              (None, 26)                858
=================================================================
Total params: 5,984
Trainable params: 5,984
Non-trainable params: 0
_________________________________________________________________
```

In [35]:
```
#Compiling the model
model3.compile(loss='sparse_categorical_crossentropy',
                      optimizer='adam',
                      metrics=['accuracy'])
```

Model 2 witout using generator:

In [36]:
```
# Training the model
#Witout generator # Z normalization

model3.fit(X_train_norm, y_train, batch_size = 100, epochs = 20)
```

```
Epoch 1/20
14000/14000 [==============================] - 1s 36us/step - loss: 2.4811 - accuracy: 0.3109
Epoch 2/20
14000/14000 [==============================] - 0s 32us/step - loss: 1.1467 - accuracy: 0.6637
Epoch 3/20
14000/14000 [==============================] - 1s 49us/step - loss: 0.8839 - accuracy: 0.7419
Epoch 4/20
14000/14000 [==============================] - 1s 44us/step - loss: 0.7581 - accuracy: 0.7707
Epoch 5/20
14000/14000 [==============================] - 0s 29us/step - loss: 0.6669 - accuracy: 0.7970
Epoch 6/20
14000/14000 [==============================] - 0s 24us/step - loss: 0.5999 - accuracy: 0.8171
Epoch 7/20
14000/14000 [==============================] - 0s 26us/step - loss: 0.5452 - accuracy: 0.8343
Epoch 8/20
14000/14000 [==============================] - 0s 26us/step - loss: 0.5007 - accuracy: 0.8480
Epoch 9/20
14000/14000 [==============================] - 0s 26us/step - loss: 0.4630 - accuracy: 0.8602
Epoch 10/20
14000/14000 [==============================] - 0s 28us/step - loss: 0.4323 - accuracy: 0.8684
Epoch 11/20
14000/14000 [==============================] - 0s 30us/step - loss: 0.4028 - accuracy: 0.8761
Epoch 12/20
14000/14000 [==============================] - 0s 30us/step - loss: 0.3797 - accuracy: 0.8869
Epoch 13/20
14000/14000 [==============================] - 0s 34us/step - loss: 0.3600 - accuracy: 0.8909
Epoch 14/20
14000/14000 [==============================] - 0s 33us/step - loss: 0.3395 - accuracy: 0.8979
Epoch 15/20
14000/14000 [==============================] - 0s 32us/step - loss: 0.3213 - accuracy: 0.9008
Epoch 16/20
14000/14000 [==============================] - 0s 33us/step - loss: 0.3091 - accuracy: 0.9035
Epoch 17/20
14000/14000 [==============================] - 1s 41us/step - loss: 0.2942 - accuracy: 0.9081
Epoch 18/20
14000/14000 [==============================] - 1s 46us/step - loss: 0.2799 - accuracy: 0.9134
Epoch 19/20
14000/14000 [==============================] - 0s 28us/step - loss: 0.2678 - accuracy: 0.9170
Epoch 20/20
14000/14000 [==============================] - 1s 46us/step - loss: 0.2543 - accuracy: 0.9219
```

Out[36]:
```
<keras.callbacks.callbacks.History at 0x1aa1d025888>
```

In [37]:
```
test_loss, test_acc = model3.evaluate(X_test_norm, y_test)

print('Test Data accuracy:', test_acc)
```

```
6000/6000 [==============================] - 0s 40us/step
Test Data accuracy: 0.9035000205039978
```

In [38]:
```
# Training the model
#Witout generator #min-max Normalization

model3.fit(X_train_minmax, y_train, batch_size = 100, epochs = 20)
```

```
Epoch 1/20
14000/14000 [==============================] - 0s 27us/step - loss: 1.5660 - accuracy: 0.6151
Epoch 2/20
14000/14000 [==============================] - 0s 26us/step - loss: 0.8927 - accuracy: 0.7622
Epoch 3/20
14000/14000 [==============================] - 0s 26us/step - loss: 0.7659 - accuracy: 0.7893
Epoch 4/20
14000/14000 [==============================] - 0s 30us/step - loss: 0.6935 - accuracy: 0.8059
Epoch 5/20
14000/14000 [==============================] - 0s 29us/step - loss: 0.6391 - accuracy: 0.8214
Epoch 6/20
14000/14000 [==============================] - 0s 31us/step - loss: 0.5996 - accuracy: 0.8269
Epoch 7/20
14000/14000 [==============================] - 0s 27us/step - loss: 0.5646 - accuracy: 0.8338
Epoch 8/20
14000/14000 [==============================] - 0s 25us/step - loss: 0.5358 - accuracy: 0.8434
Epoch 9/20
14000/14000 [==============================] - 0s 29us/step - loss: 0.5103 - accuracy: 0.8504
Epoch 10/20
14000/14000 [==============================] - 1s 43us/step - loss: 0.4844 - accuracy: 0.8578
Epoch 11/20
14000/14000 [==============================] - 1s 43us/step - loss: 0.4651 - accuracy: 0.8649
Epoch 12/20
14000/14000 [==============================] - 0s 29us/step - loss: 0.4466 - accuracy: 0.8688
Epoch 13/20
14000/14000 [==============================] - 0s 26us/step - loss: 0.4311 - accuracy: 0.8736
Epoch 14/20
14000/14000 [==============================] - 1s 41us/step - loss: 0.4161 - accuracy: 0.8769
Epoch 15/20
14000/14000 [==============================] - 0s 24us/step - loss: 0.4055 - accuracy: 0.8789
Epoch 16/20
14000/14000 [==============================] - 0s 32us/step - loss: 0.3922 - accuracy: 0.8823
Epoch 17/20
14000/14000 [==============================] - 0s 28us/step - loss: 0.3790 - accuracy: 0.8871
Epoch 18/20
14000/14000 [==============================] - 0s 23us/step - loss: 0.3687 - accuracy: 0.8908
Epoch 19/20
14000/14000 [==============================] - 0s 28us/step - loss: 0.3585 - accuracy: 0.8926
Epoch 20/20
14000/14000 [==============================] - 0s 28us/step - loss: 0.3538 - accuracy: 0.8959
```

Out[38]:
```
<keras.callbacks.callbacks.History at 0x1aa1d431788>
```

In [39]:
```
test_loss, test_acc = model3.evaluate(X_test_minmax, y_test)

print('Test Data accuracy:', test_acc)
```

```
6000/6000 [==============================] - 0s 29us/step
Test Data accuracy: 0.8606666922569275
```

Model 2 with generator 1:

In [40]:
```
#Training the model
#With generator 1 # Z normalization

model3.fit_generator(train_gen11, steps_per_epoch = 21, epochs = 20)
```

```
Epoch 1/20
21/21 [==============================] - 0s 6ms/step - loss: 2.2216 - accuracy: 0.7609
Epoch 2/20
21/21 [==============================] - 0s 7ms/step - loss: 0.7764 - accuracy: 0.8588
Epoch 3/20
21/21 [==============================] - 0s 7ms/step - loss: 0.5665 - accuracy: 0.8837
Epoch 4/20
21/21 [==============================] - 0s 7ms/step - loss: 0.4765 - accuracy: 0.8927
Epoch 5/20
21/21 [==============================] - 0s 7ms/step - loss: 0.4353 - accuracy: 0.9001
Epoch 6/20
21/21 [==============================] - 0s 7ms/step - loss: 0.3935 - accuracy: 0.9036
Epoch 7/20
21/21 [==============================] - 0s 8ms/step - loss: 0.3742 - accuracy: 0.9070
Epoch 8/20
21/21 [==============================] - 0s 6ms/step - loss: 0.3462 - accuracy: 0.9101
Epoch 9/20
21/21 [==============================] - 0s 7ms/step - loss: 0.3349 - accuracy: 0.9131
Epoch 10/20
21/21 [==============================] - 0s 7ms/step - loss: 0.3138 - accuracy: 0.9152
Epoch 11/20
21/21 [==============================] - 0s 7ms/step - loss: 0.3066 - accuracy: 0.9175
Epoch 12/20
21/21 [==============================] - 0s 7ms/step - loss: 0.2893 - accuracy: 0.9199
Epoch 13/20
21/21 [==============================] - 0s 7ms/step - loss: 0.2852 - accuracy: 0.9212
Epoch 14/20
21/21 [==============================] - 0s 7ms/step - loss: 0.2698 - accuracy: 0.9233
Epoch 15/20
21/21 [==============================] - 0s 6ms/step - loss: 0.2678 - accuracy: 0.9242
Epoch 16/20
21/21 [==============================] - 0s 7ms/step - loss: 0.2536 - accuracy: 0.9264: 0s - los
s: 0.2609 - accuracy: 0.92
Epoch 17/20
21/21 [==============================] - 0s 7ms/step - loss: 0.2532 - accuracy: 0.9268
Epoch 18/20
21/21 [==============================] - 0s 6ms/step - loss: 0.2401 - accuracy: 0.9297
Epoch 19/20
21/21 [==============================] - 0s 7ms/step - loss: 0.2408 - accuracy: 0.9293
Epoch 20/20
21/21 [==============================] - 0s 7ms/step - loss: 0.2285 - accuracy: 0.9318
```

Out[40]:
```
<keras.callbacks.callbacks.History at 0x1aa1d438608>
```

In [41]:
```
test_loss, test_acc = model3.evaluate_generator(test_gen11, steps = 18)

print('Test Data accuracy:', test_acc)
```

```
Test Data accuracy: 0.906000018119812
```

In [42]:
```
#Training the model
#With generator 1 # min-max normalization

model3.fit_generator(train_gen12, steps_per_epoch = 21, epochs = 20)
```

```
Epoch 1/20
21/21 [==============================] - 0s 5ms/step - loss: 2.1425 - accuracy: 0.5429
Epoch 2/20
21/21 [==============================] - 0s 7ms/step - loss: 0.6404 - accuracy: 0.8004
Epoch 3/20
21/21 [==============================] - 0s 7ms/step - loss: 0.4763 - accuracy: 0.8728
Epoch 4/20
21/21 [==============================] - 0s 7ms/step - loss: 0.4313 - accuracy: 0.8836
Epoch 5/20
21/21 [==============================] - 0s 6ms/step - loss: 0.4084 - accuracy: 0.8893
Epoch 6/20
21/21 [==============================] - 0s 6ms/step - loss: 0.3950 - accuracy: 0.8925
Epoch 7/20
21/21 [==============================] - 0s 6ms/step - loss: 0.3852 - accuracy: 0.8948
Epoch 8/20
21/21 [==============================] - 0s 7ms/step - loss: 0.3766 - accuracy: 0.8982
Epoch 9/20
21/21 [==============================] - 0s 6ms/step - loss: 0.3710 - accuracy: 0.8998
Epoch 10/20
21/21 [==============================] - 0s 7ms/step - loss: 0.3641 - accuracy: 0.9017
Epoch 11/20
21/21 [==============================] - 0s 5ms/step - loss: 0.3607 - accuracy: 0.9018
Epoch 12/20
21/21 [==============================] - 0s 6ms/step - loss: 0.3546 - accuracy: 0.9047
Epoch 13/20
21/21 [==============================] - 0s 7ms/step - loss: 0.3525 - accuracy: 0.9042
Epoch 14/20
21/21 [==============================] - 0s 7ms/step - loss: 0.3467 - accuracy: 0.9070
Epoch 15/20
21/21 [==============================] - 0s 7ms/step - loss: 0.3454 - accuracy: 0.9059
Epoch 16/20
21/21 [==============================] - 0s 7ms/step - loss: 0.3397 - accuracy: 0.9090
Epoch 17/20
21/21 [==============================] - 0s 7ms/step - loss: 0.3389 - accuracy: 0.9080
Epoch 18/20
21/21 [==============================] - 0s 6ms/step - loss: 0.3334 - accuracy: 0.9105
Epoch 19/20
21/21 [==============================] - 0s 7ms/step - loss: 0.3330 - accuracy: 0.9090
Epoch 20/20
21/21 [==============================] - 0s 7ms/step - loss: 0.3279 - accuracy: 0.9115
```

Out[42]:
```
<keras.callbacks.callbacks.History at 0x1aa1d440688>
```

In [43]:
```
test_loss, test_acc = model3.evaluate_generator(test_gen12, steps = 18)

print('Test Data accuracy:', test_acc)
```

```
Test Data accuracy: 0.8691666722297668
```

Model 2 with generator 2:

In [44]:
```
#Training the model
#With generator 2 # Z normalization

model3.fit_generator(train_gen21, steps_per_epoch = 21, epochs = 20)
```

```
Epoch 1/20
21/21 [==============================] - 1s 57ms/step - loss: 7.0986 - accuracy: 0.4918 0s - los
s: 7.1974 - accuracy: 0.49
Epoch 2/20
21/21 [==============================] - 1s 42ms/step - loss: 4.0823 - accuracy: 0.4984
Epoch 3/20
21/21 [==============================] - 1s 42ms/step - loss: 2.8847 - accuracy: 0.4946
Epoch 4/20
21/21 [==============================] - 1s 39ms/step - loss: 2.3553 - accuracy: 0.4922 0s - los
s: 2.3989 - accuracy
Epoch 5/20
21/21 [==============================] - 1s 54ms/step - loss: 2.0958 - accuracy: 0.4834
Epoch 6/20
21/21 [==============================] - 1s 46ms/step - loss: 1.9090 - accuracy: 0.4920
Epoch 7/20
21/21 [==============================] - 1s 37ms/step - loss: 1.8047 - accuracy: 0.4932
Epoch 8/20
21/21 [==============================] - 1s 46ms/step - loss: 1.7546 - accuracy: 0.4993
Epoch 9/20
21/21 [==============================] - 1s 42ms/step - loss: 1.7023 - accuracy: 0.4989 0s - los
s: 1.7123 - accuracy
Epoch 10/20
21/21 [==============================] - 1s 42ms/step - loss: 1.6438 - accuracy: 0.5137
Epoch 11/20
21/21 [==============================] - 1s 50ms/step - loss: 1.6155 - accuracy: 0.5153
Epoch 12/20
21/21 [==============================] - 1s 35ms/step - loss: 1.5917 - accuracy: 0.5184
Epoch 13/20
21/21 [==============================] - 1s 38ms/step - loss: 1.5618 - accuracy: 0.5204
Epoch 14/20
21/21 [==============================] - 1s 40ms/step - loss: 1.5556 - accuracy: 0.5240 0s - los
s: 1.5572 - accu
Epoch 15/20
21/21 [==============================] - 1s 33ms/step - loss: 1.5308 - accuracy: 0.5263
Epoch 16/20
21/21 [==============================] - 1s 40ms/step - loss: 1.5204 - accuracy: 0.5287
Epoch 17/20
21/21 [==============================] - 1s 47ms/step - loss: 1.5127 - accuracy: 0.5335
Epoch 18/20
21/21 [==============================] - 1s 40ms/step - loss: 1.4940 - accuracy: 0.5384
Epoch 19/20
21/21 [==============================] - 1s 45ms/step - loss: 1.4816 - accuracy: 0.5354 0s - los
s: 1
Epoch 20/20
21/21 [==============================] - 1s 58ms/step - loss: 1.4660 - accuracy: 0.5369
```

Out[44]:
```
<keras.callbacks.callbacks.History at 0x1aa1d438508>
```

In [45]:
```
test_loss, test_acc = model3.evaluate_generator(test_gen21, steps = 18)

print('Test Data accuracy:', test_acc)
```

```
Test Data accuracy: 0.5399444699287415
```

In [46]:
```
#Training the model
#With generator 2 # min-max normalization

model3.fit_generator(train_gen21, steps_per_epoch = 21, epochs = 20)
```

```
Epoch 1/20
21/21 [==============================] - 1s 47ms/step - loss: 1.4548 - accuracy: 0.5404
Epoch 2/20
21/21 [==============================] - 1s 49ms/step - loss: 1.4516 - accuracy: 0.5490 0s - los
s: 1.4479 - accura
Epoch 3/20
21/21 [==============================] - 1s 46ms/step - loss: 1.4393 - accuracy: 0.5435
Epoch 4/20
21/21 [==============================] - 1s 42ms/step - loss: 1.4420 - accuracy: 0.5448
Epoch 5/20
21/21 [==============================] - 1s 40ms/step - loss: 1.4161 - accuracy: 0.5534
Epoch 6/20
21/21 [==============================] - 1s 44ms/step - loss: 1.4160 - accuracy: 0.5495
Epoch 7/20
21/21 [==============================] - 1s 54ms/step - loss: 1.4098 - accuracy: 0.5527
Epoch 8/20
21/21 [==============================] - 1s 35ms/step - loss: 1.4070 - accuracy: 0.5561
Epoch 9/20
21/21 [==============================] - 1s 42ms/step - loss: 1.3852 - accuracy: 0.5593 0s - los
s: 1.3931 - accu
Epoch 10/20
21/21 [==============================] - 1s 54ms/step - loss: 1.3734 - accuracy: 0.5595 0s - los
s: 1.3740 - accuracy:
Epoch 11/20
21/21 [==============================] - 1s 55ms/step - loss: 1.3665 - accuracy: 0.5610
Epoch 12/20
21/21 [==============================] - 1s 47ms/step - loss: 1.3801 - accuracy: 0.5621
Epoch 13/20
21/21 [==============================] - 1s 50ms/step - loss: 1.3712 - accuracy: 0.5584
Epoch 14/20
21/21 [==============================] - 1s 40ms/step - loss: 1.3525 - accuracy: 0.5667 0s - los
s: 1.3496 -
Epoch 15/20
21/21 [==============================] - 1s 39ms/step - loss: 1.3725 - accuracy: 0.5621
Epoch 16/20
21/21 [==============================] - 1s 42ms/step - loss: 1.3621 - accuracy: 0.5675
Epoch 17/20
21/21 [==============================] - 1s 31ms/step - loss: 1.3594 - accuracy: 0.5689 0s - los
s: 1.3545 - accuracy:
Epoch 18/20
21/21 [==============================] - 1s 43ms/step - loss: 1.3364 - accuracy: 0.5669
Epoch 19/20
21/21 [==============================] - 1s 37ms/step - loss: 1.3297 - accuracy: 0.5685
Epoch 20/20
21/21 [==============================] - 1s 38ms/step - loss: 1.3367 - accuracy: 0.5706 0s - los
s: 1.3356 - accuracy:
```

Out[46]:
```
<keras.callbacks.callbacks.History at 0x1aa1ac65888>
```

In [91]:
```
test_loss, test_acc = model3.evaluate_generator(test_gen22, steps = 18)

print('Test Data accuracy:', test_acc)
```

```
Test Data accuracy: 0.08116666972637177
```

Models without the addition noise has higher accuracy of classifying than the onw with noise.
Z normalization is better than min-max normalization for this dataset.