Deep Learning Report: Classification of Pistachios with Neural Networks

Author: Syed Bokhari

Date of Submission: 30th December 2022

Introduction:

The dataset used for this project is from Kaggle. The project from which the data comes was originally based on classifying images of different varieties of pistachios. The datasets used in this project were collected from these images using feature extraction by the original team. Originally, 28 features were extracted from the images of pistachios. Subsequently, some features were dropped to reduce the number of features to 16 in a separate dataset. The original datasets can be found at: https://www.kaggle.com/datasets/muratkokludataset/pistachio-image-dataset

Both 16-feature and 28-feature datasets are used in the project. The features in the 28-feature dataset capture various morphological characteristics, shape features and color aspects of two different pistachio species from high-resolution images. The 16-feature dataset drops all the color aspects but keeps the morphological and shape features.

The details of the different features are given below:

Morphological Features (12 Features)

1. **Area (int):** Area covered by the object in the image. It roughly corresponds to the number of pixels in the region covered by the object.
2. **Perimeter (float):** Perimeter pixels covered by the object in the image.
3. **Major_Axis (float):** The longest diameter of the object in the image. (Pistachios are elliptical in 2D images.)
4. **Minor_Axis (float):** The shortest diameter of the object in the image.
5. **Eccentricity (float):** The ratio of the distance between the foci of the object in the image and its major axis.
6. **Eqdiasq (float):** Equivalent diameter squared of the object in the image.
7. Solidity (float): The Area of the image divided by its Convex Area. (Area/ConvexArea)
8. **Convex_Area (int):** The convex hull of the region occupied by the object in the image. It is greater than or equal to the Area.
9. **Extent (float):** Area of the object in the image divided by the area of its bounding rectangle.
10. **Aspect_Ratio (float):** Proportional relationship between an image's width and height.
11. **Roundness (float):** The ratio of the surface area of the object to the area of the circle whose diameter is equal to the maximum diameter of the object
12. **Compactness (float):** The ratio of the area of an object to the area of a circle with the same perimeter.

Shape Features (4 Features)

13. **Shapefactor_1 (float):** First specific shape feature.
14. **Shapefactor_2 (float):** Second specific shape feature.
15. **Shapefactor_3 (float):** Third specific shape feature.
16. **Shapefactor_4 (float):** Fourth specific shape feature.

Color Features (12 Features)

17. **Mean_RR (float):** Mean of Red-Red color component.
18. **Mean_RG (float):** Mean of Red-Green color component.
19. **Mean_RB (float):** Mean of Red-Blue color component.
20. **StdDev_RR (float):** Standard Deviation of Red-Red color component.
21. **StdDev_RG (float):** Standard Deviation of Red- Green color component.
22. **StdDev_RB (float):** Standard Deviation of Red-Blue color component.
23. **Skew_RR (float):** Skew of Red-Red color component.
24. **Skew_RG (float):** Skew of Red- Green color component.
25. **Skew_RB (float):** Skew of Red-Blue color component.
26. **Kurtosis_RR (float):** Kurtosis of Red-Red color component.
27. **Kurtosis_RG (float):** Kurtosis of Red- Green color component.
28. **Kurtosis_RB (float):** Kurtosis of Red-Blue color component.

Class (object): A string column representing the class of the data point.

Objectives:

The aim of this project is to investigate the usage of neural networks and deep learning for classification problems. The project makes use of neural networks with different number of hidden layers for classification as well as a baseline classification technique and compares the results. Additionally, the project will use two datasets: a 28-feature original dataset and a reduced copy of the original, a 16-feature dataset. The results of both these datasets will be compared for classification to determine the impact of data size and dimensions on deep learning as well. The primary focus of the project will be on evaluating the results from classification to better understand deep learning and the impact of data on it. Therefore, interpretation of the results will be important and is discussed in the Results section. The insights derived from the Results section will be presented in the Key Findings section. Steps for future efforts to improve this analysis are discussed in the Future Steps section.

Data Exploration:

The datasets were initially in xlsx format but were loaded in Pandas dataframes. Pandas is a very useful library in Python for data preprocessing. It chiefly makes use of Dataframes, a built-in data structure, for performing data preprocessing and analysis in an efficient manner.

```
df_16 = pd.read_excel('Pistachio_16_Features_Dataset.xls')
df_16.head()
```

	AREA	PERIMETER	MAJOR_AXIS	MINOR_AXIS	ECCENTRICITY	EQDIASQ	SOLIDITY	CONVEX_AREA	EXTENT	ASPECT_RATIO	ROUNDNESS	COMPACT
0	63391	1568.405	390.3396	236.7461	0.7951	284.0984	0.8665	73160	0.6394	1.6488	0.3238	(
1	68358	1942.187	410.8594	234.7525	0.8207	295.0188	0.8765	77991	0.6772	1.7502	0.2277	(
2	73589	1246.538	452.3630	220.5547	0.8731	306.0987	0.9172	80234	0.7127	2.0510	0.5951	(
3	71106	1445.261	429.5291	216.0765	0.8643	300.8903	0.9589	74153	0.7028	1.9879	0.4278	(
4	80087	1251.524	469.3783	220.9344	0.8823	319.3273	0.9657	82929	0.7459	2.1245	0.6425	(

```
df_28 = pd.read_excel('Pistachio_28_Features_Dataset.xls')
df_28.head()
```

	Area	Perimeter	Major_Axis	Minor_Axis	Eccentricity	Eqdiasq	Solidity	Convex_Area	Extent	Aspect_Ratio	...	StdDev_RR	StdDev_RG	StdDev_RB
0	63391	1568.405	390.3396	236.7461	0.7951	284.0984	0.8665	73160	0.6394	1.6488	...	17.7206	19.6024	21.1342
1	68358	1942.187	410.8594	234.7525	0.8207	295.0188	0.8765	77991	0.6772	1.7502	...	26.7061	27.2112	25.1035
2	73589	1246.538	452.3630	220.5547	0.8731	306.0987	0.9172	80234	0.7127	2.0510	...	19.0129	20.0703	20.7006
3	71106	1445.261	429.5291	216.0765	0.8643	300.8903	0.9589	74153	0.7028	1.9879	...	18.1773	18.7152	29.7883
4	80087	1251.524	469.3783	220.9344	0.8823	319.3273	0.9657	82929	0.7459	2.1245	...	23.4298	24.0878	23.1157

5 rows × 29 columns

After loading the datasets in dataframes, basic data exploration was undertaken.

```
df_16.shape
```

```
(2148, 17)
```

```
df_28.shape
```

```
(2148, 29)
```

All columns except the class column were in numeric form for both datasets which is very convenient.

```python
df_16.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2148 entries, 0 to 2147
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   AREA            2148 non-null   int64  
 1   PERIMETER       2148 non-null   float64
 2   MAJOR_AXIS      2148 non-null   float64
 3   MINOR_AXIS      2148 non-null   float64
 4   ECCENTRICITY    2148 non-null   float64
 5   EQDIASQ         2148 non-null   float64
 6   SOLIDITY        2148 non-null   float64
 7   CONVEX_AREA     2148 non-null   int64  
 8   EXTENT          2148 non-null   float64
 9   ASPECT_RATIO    2148 non-null   float64
 10  ROUNDNESS       2148 non-null   float64
 11  COMPACTNESS     2148 non-null   float64
 12  SHAPEFACTOR_1   2148 non-null   float64
 13  SHAPEFACTOR_2   2148 non-null   float64
 14  SHAPEFACTOR_3   2148 non-null   float64
 15  SHAPEFACTOR_4   2148 non-null   float64
 16  Class           2148 non-null   object 
dtypes: float64(14), int64(2), object(1)
memory usage: 285.4+ KB
```

```
df_28.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2148 entries, 0 to 2147
Data columns (total 29 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Area            2148 non-null   int64  
 1   Perimeter       2148 non-null   float64
 2   Major_Axis      2148 non-null   float64
 3   Minor_Axis      2148 non-null   float64
 4   Eccentricity    2148 non-null   float64
 5   Eqdiasq         2148 non-null   float64
 6   Solidity        2148 non-null   float64
 7   Convex_Area     2148 non-null   int64  
 8   Extent          2148 non-null   float64
 9   Aspect_Ratio    2148 non-null   float64
 10  Roundness       2148 non-null   float64
 11  Compactness     2148 non-null   float64
 12  Shapefactor_1   2148 non-null   float64
 13  Shapefactor_2   2148 non-null   float64
 14  Shapefactor_3   2148 non-null   float64
 15  Shapefactor_4   2148 non-null   float64
 16  Mean_RR         2148 non-null   float64
 17  Mean_RG         2148 non-null   float64
 18  Mean_RB         2148 non-null   float64
 19  StdDev_RR       2148 non-null   float64
 20  StdDev_RG       2148 non-null   float64
 21  StdDev_RB       2148 non-null   float64
 22  Skew_RR         2148 non-null   float64
 23  Skew_RG         2148 non-null   float64
 24  Skew_RB         2148 non-null   float64
 25  Kurtosis_RR     2148 non-null   float64
 26  Kurtosis_RG     2148 non-null   float64
 27  Kurtosis_RB     2148 non-null   float64
 28  Class           2148 non-null   object 
dtypes: float64(26), int64(2), object(1)
memory usage: 486.8+ KB
```

The features however, were not in standardized form, with a varying range of values. This would require handling at later stages as Neural Network models are sensitive to differences in data range between features in calculating weights for calculation purposes.

```
df_16.describe()
```

	AREA	PERIMETER	MAJOR_AXIS	MINOR_AXIS	ECCENTRICITY	EQDIASQ	SOLIDITY	CONVEX_AREA	EXTENT	ASPECT_RATIO	RC
count	2148.000000	2148.000000	2148.000000	2148.000000	2148.000000	2148.000000	2148.000000	2148.000000	2148.000000	2148.000000	2
mean	79950.655493	1425.971751	446.248968	238.311842	0.840219	317.919173	0.940093	85015.839851	0.716067	1.898154	
std	13121.737799	375.565503	32.445304	30.310695	0.048759	26.908600	0.050452	13154.919327	0.052532	0.240100	
min	29808.000000	858.363000	320.344500	133.509600	0.504900	194.814600	0.588000	37935.000000	0.427200	1.158500	
25%	71936.750000	1170.996250	426.508750	217.875825	0.817500	302.642850	0.919850	76467.000000	0.687000	1.736375	
50%	79905.500000	1262.785500	448.574750	236.416350	0.849650	318.965300	0.954150	85075.500000	0.726500	1.896250	
75%	89030.500000	1607.906250	468.509400	257.760150	0.875200	336.685525	0.976925	93893.500000	0.753600	2.067025	
max	124008.000000	2755.049100	541.966100	383.046100	0.946000	397.356100	0.995100	132478.000000	0.820400	3.085800	

```
df_28.describe()
```

	Area	Perimeter	Major_Axis	Minor_Axis	Eccentricity	Eqdiasq	Solidity	Convex_Area	Extent	Aspect_Ratio	...	Me
count	2148.000000	2148.000000	2148.000000	2148.000000	2148.000000	2148.000000	2148.000000	2148.000000	2148.000000	2148.000000	...	2148.0
mean	79950.655493	1425.971751	446.248968	238.311842	0.840219	317.919173	0.940093	85015.839851	0.716067	1.898154	...	191.9
std	13121.737799	375.565503	32.445304	30.310695	0.048759	26.908600	0.050452	13154.919327	0.052532	0.240100	...	13.0
min	29808.000000	858.363000	320.344500	133.509600	0.504900	194.814600	0.588000	37935.000000	0.427200	1.158500	...	146.7
25%	71936.750000	1170.996250	426.508750	217.875825	0.817500	302.642850	0.919850	76467.000000	0.687000	1.736375	...	182.9
50%	79905.500000	1262.785500	448.574750	236.416350	0.849650	318.965300	0.954150	85075.500000	0.726500	1.896250	...	192.0
75%	89030.500000	1607.906250	468.509400	257.760150	0.875200	336.685525	0.976925	93893.500000	0.753600	2.067025	...	201.0
max	124008.000000	2755.049100	541.966100	383.046100	0.946000	397.356100	0.995100	132478.000000	0.820400	3.085800	...	235.0

8 rows × 28 columns

The dataset does not have any missing values which is also quite convenient.

```
(df_16.isna()).sum()
```

```
AREA             0
PERIMETER        0
MAJOR_AXIS       0
MINOR_AXIS       0
ECCENTRICITY     0
EQDIASQ          0
SOLIDITY         0
CONVEX_AREA      0
EXTENT           0
ASPECT_RATIO     0
ROUNDNESS        0
COMPACTNESS      0
SHAPEFACTOR_1    0
SHAPEFACTOR_2    0
SHAPEFACTOR_3    0
SHAPEFACTOR_4    0
Class            0
dtype: int64
```

```
(df_28.isna()).sum()
```

```
Area             0
Perimeter        0
Major_Axis       0
Minor_Axis       0
Eccentricity     0
Eqdiasq          0
Solidity         0
Convex_Area      0
Extent           0
Aspect_Ratio     0
Roundness        0
Compactness      0
Shapefactor_1    0
Shapefactor_2    0
Shapefactor_3    0
Shapefactor_4    0
Mean_RR          0
Mean_RG          0
Mean_RB          0
StdDev_RR        0
StdDev_RG        0
StdDev_RB        0
Skew_RR          0
Skew_RG          0
Skew_RB          0
Kurtosis_RR      0
Kurtosis_RG      0
Kurtosis_RB      0
Class            0
dtype: int64
```

The distribution of values in the class column was also checked.

```
df_16['Class'].value_counts(normalize=True)
```

```
Kirmizi_Pistachio    0.573557
Siit_Pistachio       0.426443
Name: Class, dtype: float64
```

```
df_28['Class'].value_counts(normalize=True)
```

```
Kirmizi_Pistachio    0.573557
Siirt_Pistachio      0.426443
Name: Class, dtype: float64
```

The data in the class column while not evenly distributed between the two classes, was close enough to not warrant any interventions.

Plan:

The aim of the project was to make use of Neural Networks for classification. Initially, the data will be prepared for classification with some feature engineering to create the features and targets for both datasets. Afterwards, both datasets will be split in training and testing datasets.

After the training and testing data has been prepared for both datasets, a Random Forest model will be used as a baseline classifier and its results recorded for both datasets. The datasets will then be standardized as Neural Networks are very sensitive to variations in data range. Subsequently, classification will be performed with a 0-hidden layer Neural Network, a 1-hidden layer Neural Network, 2-hidden layers Neural Network and 3-hidden layers Neural Network on both datasets. All these results will also be recorded and the key findings from these will be elaborated upon.

Feature Engineering:

The classes in the datasets needed to be encoded in numeric format. As only two classes were there, they could easily be converted into binary format. During the process of label encoding the Class column, the target datasets were created also.

```
y_16 = df_16['Class']
df_16['Target'] = [1 if y == 'Kirmizi_Pistachio' else 0 for y in y_16]
df_16.head()
```

	AREA	PERIMETER	MAJOR_AXIS	MINOR_AXIS	ECCENTRICITY	EQDIASQ	SOLIDITY	CONVEX_AREA	EXTENT	ASPECT_RATIO	ROUNDNESS	COMPACT
0	63391	1568.405	390.3396	236.7461	0.7951	284.0984	0.8665	73160	0.6394	1.6488	0.3238	(
1	68358	1942.187	410.8594	234.7525	0.8207	295.0188	0.8765	77991	0.6772	1.7502	0.2277	(
2	73589	1246.538	452.3630	220.5547	0.8731	306.0987	0.9172	80234	0.7127	2.0510	0.5951	(
3	71106	1445.261	429.5291	216.0765	0.8643	300.8903	0.9589	74153	0.7028	1.9879	0.4278	(
4	80087	1251.524	469.3783	220.9344	0.8823	319.3273	0.9657	82929	0.7459	2.1245	0.6425	(

```
df_16['Target'].value_counts(normalize=True)
```

```
1    0.573557
0    0.426443
Name: Target, dtype: float64
```

```
y_16 = df_16['Target']
y_16.head()
```

```
0    1
1    1
2    1
3    1
4    1
Name: Target, dtype: int64
```

```
y_28 = df_28['Class']
df_28['Target'] = [1 if y == 'Kirmizi_Pistachio' else 0 for y in y_28]
df_28.head()
```

	Area	Perimeter	Major_Axis	Minor_Axis	Eccentricity	Eqdiasq	Solidity	Convex_Area	Extent	Aspect_Ratio	...	StdDev_RG	StdDev_RB	Skew_RR	S
0	63391	1568.405	390.3396	236.7461	0.7951	284.0984	0.8665	73160	0.6394	1.6488	...	19.6024	21.1342	0.4581	
1	68358	1942.187	410.8594	234.7525	0.8207	295.0188	0.8765	77991	0.6772	1.7502	...	27.2112	25.1035	-0.3847	
2	73589	1246.538	452.3630	220.5547	0.8731	306.0987	0.9172	80234	0.7127	2.0510	...	20.0703	20.7006	-0.6014	
3	71106	1445.261	429.5291	216.0765	0.8643	300.8903	0.9589	74153	0.7028	1.9879	...	18.7152	29.7883	-0.6943	
4	80087	1251.524	469.3783	220.9344	0.8823	319.3273	0.9657	82929	0.7459	2.1245	...	24.0878	23.1157	-0.9287	

5 rows × 30 columns

```
df_28['Target'].value_counts(normalize=True)
```

```
1    0.573557
0    0.426443
Name: Target, dtype: float64
```

```
y_28 = df_28['Target']
y_28.head()
```

```
0    1
1    1
2    1
3    1
4    1
Name: Target, dtype: int64
```

Subsequently, datasets were used to create the features sets as well.

```
X_16 = df_16.drop('Class', axis=1)
X_16.shape
```

```
(2148, 16)
```

```
X_16.head()
```

	AREA	PERIMETER	MAJOR_AXIS	MINOR_AXIS	ECCENTRICITY	EQDIASQ	SOLIDITY	CONVEX_AREA	EXTENT	ASPECT_RATIO	ROUNDNESS	COMPACT
0	63391	1568.405	390.3396	236.7461	0.7951	284.0984	0.8665	73160	0.6394	1.6488	0.3238	(
1	68358	1942.187	410.8594	234.7525	0.8207	295.0188	0.8765	77991	0.6772	1.7502	0.2277	(
2	73589	1246.538	452.3630	220.5547	0.8731	306.0987	0.9172	80234	0.7127	2.0510	0.5951	(
3	71106	1445.261	429.5291	216.0765	0.8643	300.8903	0.9589	74153	0.7028	1.9879	0.4278	(
4	80087	1251.524	469.3783	220.9344	0.8823	319.3273	0.9657	82929	0.7459	2.1245	0.6425	(

```
X_28 = df_28.drop('Class', axis=1)
X_28.shape
```

```
(2148, 28)
```

```
X_28.head()
```

	Area	Perimeter	Major_Axis	Minor_Axis	Eccentricity	Eqdiasq	Solidity	Convex_Area	Extent	Aspect_Ratio	...	Mean_RB	StdDev_RR	StdDev_RG	S
0	63391	1568.405	390.3396	236.7461	0.7951	284.0984	0.8665	73160	0.6394	1.6488	...	165.3167	17.7206	19.6024	
1	68358	1942.187	410.8594	234.7525	0.8207	295.0188	0.8765	77991	0.6772	1.7502	...	187.3744	26.7061	27.2112	
2	73589	1246.538	452.3630	220.5547	0.8731	306.0987	0.9172	80234	0.7127	2.0510	...	187.7118	19.0129	20.0703	
3	71106	1445.261	429.5291	216.0765	0.8643	300.8903	0.9589	74153	0.7028	1.9879	...	187.9537	18.1773	18.7152	
4	80087	1251.524	469.3783	220.9344	0.8823	319.3273	0.9657	82929	0.7459	2.1245	...	194.4906	23.4298	24.0878	

5 rows × 28 columns

Train-Test Split:

The features datasets and the target datasets were subsequently split into separate training and testing datasets. The balance of classes in the split datasets was not seen as a cause of concern.

```
X_16_train, X_16_test, y_16_train, y_16_test = train_test_split(X_16, y_16, test_size=0.25, random_state=1)
y_16_train.value_counts(normalize=True)
```

```
1    0.572315
0    0.427685
Name: Target, dtype: float64
```

```
y_16_test.value_counts(normalize=True)
```

```
1    0.577281
0    0.422719
Name: Target, dtype: float64
```

```
X_28_train, X_28_test, y_28_train, y_28_test = train_test_split(X_28, y_28, test_size=0.25, random_state=4)
y_28_train.value_counts(normalize=True)
```

```
1    0.568591
0    0.431409
Name: Target, dtype: float64
```

```
y_28_test.value_counts(normalize=True)
```

```
1    0.588454
0    0.411546
Name: Target, dtype: float64
```

Standardization:

The final step before proceeding with classification was to standardize the data. As noted previously, some features in the dataset had widely varying values. This would cause issues with Neural Networks which might assign weights erroneously by prioritizing features with larger variations. Therefore, to avoid this problem, the features in the training data were standardized using a StandardScaler() object with the fit_transform method. Afterwards, using the calculated mean and standard deviation when standardizing the training dataset, the testing features data was standardized using the transform method. This was done for both datasets.

16 Features Dataset

```
ss_16 = StandardScaler()
X_16_train_ss = ss_16.fit_transform(X_16_train)
X_16_test_ss = ss_16.transform(X_16_test)
```

28 Features Dataset

```
ss_28 = StandardScaler()
X_28_train_ss = ss_28.fit_transform(X_28_train)
X_28_test_ss = ss_28.transform(X_28_test)
```

Deep Learning:

With the datasets ready, the next step was classification. Initially, a baseline classifier was used to establish a baseline accuracy score for both datasets. Afterwards, multiple Neural Networks with varying numbers of hidden layers were trained with the training dataset and subsequently evaluated on the testing dataset.

The Neural Networks were evaluated using the accuracy score of their predictions. These were plotted on a graph for each Neural Network for easy visualization of the results.

The accuracy score reflects the number of correctly identified labels from all present instances of a class in the data. Although not a perfect measure, it is sufficient for the purposes of this project.

Baseline Model:

The Baseline Model used in the project is a Random Forest classifier. It was set to use 200 estimators for both datasets. The baseline classifier was fitted on the training data and used to make predictions for the testing data. This was done on both datasets.

```python
rf_16 = RandomForestClassifier(n_estimators=200)
rf_16.fit(X_16_train, y_16_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```python
y_16_pred = rf_16.predict(X_16_test)
acc_16 = accuracy_score(y_16_test, y_16_pred)
print('The accuracy is: {:.3f}'.format(acc_16))
```

```
The accuracy is: 0.857
```

```python
rf_28 = RandomForestClassifier(n_estimators=200)
rf_28.fit(X_28_train, y_28_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```python
y_28_pred = rf_28.predict(X_28_test)
acc_28 = accuracy_score(y_28_test, y_28_pred)
print('The accuracy is: {:.3f}'.format(acc_28))
```

```
The accuracy is: 0.892
```

The resulting values were used as baseline values for the subsequent analysis. Note that the 16-feature dataset has a lower accuracy value than the 28-feature dataset. This is intuitively correct as more features should lead to better predictions. However, the difference in accuracy is not much between the two: 85.7 for 16-feature dataset and 89.2 for 28-feature dataset.

0-Hidden Layer Neural Network:

After the baseline model, the core of the project was approached. In the first instance, a simple Neural Network with 0 Hidden Layers was defined. The Neural Network used a relu (Rectified Linear Unit) function for the input and a sigmoid activation function for the output. It had 28 filters for the input layer. A summary of the resulting models was displayed.

For the 16-Feature set, the Neural Network had 505 trainable parameters. For the 28-Feature set, the Neural Network had a total of 841 trainable parameters. Neural Networks for both datasets were compiled using the ADAM (ADaptive Moment Estimation) optimizer. Binary cross entropy was used to estimate their accuracy.

The models for both datasets were run for 100 epochs. The accuracy for training data and testing data for each epoch (iteration in this context) was plotted on graphs for better visualization.

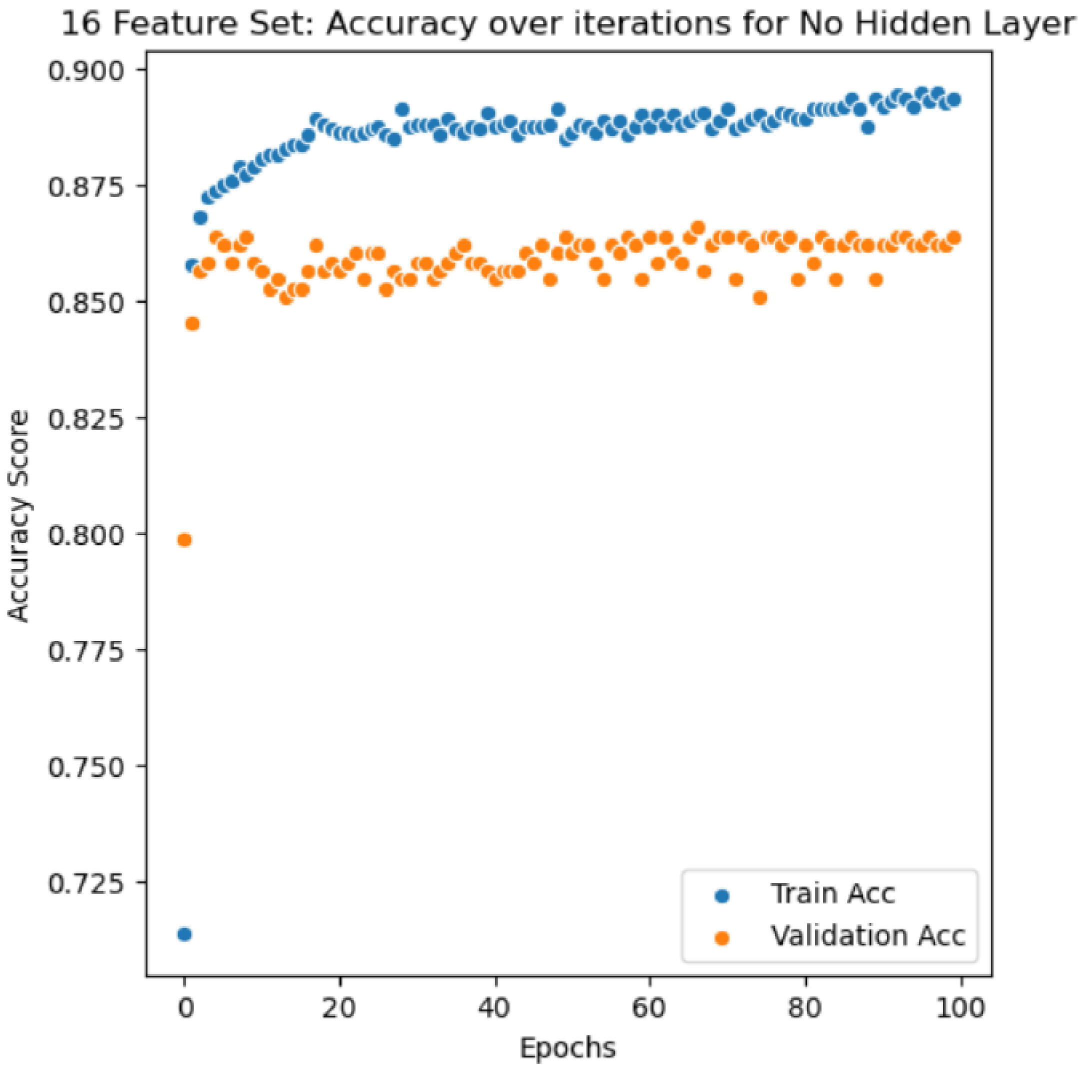
```
NN_1_16 = Sequential()
NN_1_16.add(Dense(28, input_shape = (16,), activation = 'relu'))
NN_1_16.add(Dense(1, activation='sigmoid'))
NN_1_16.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 28)                476
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 29
=================================================================
Total params: 505
Trainable params: 505
Non-trainable params: 0
_________________________________________________________________
```

```
NN_1_16.compile(Adam(lr = .001), "binary_crossentropy", metrics=["accuracy"])
run_hist_16_1 = NN_1_16.fit(X_16_train_ss, y_16_train, validation_data=(X_16_test_ss, y_16_test), epochs=100)
```

```
n = len(run_hist_16_1.history["val_acc"])
fig = plt.figure(figsize=(12, 6))

ax = fig.add_subplot(1, 2, 2)
sns.scatterplot(x=range(n), y=(run_hist_16_1.history["acc"]), ax=ax, label="Train Acc")
sns.scatterplot(x=range(n), y=(run_hist_16_1.history["val_acc"]), ax=ax, label="Validation Acc")
ax.set_xlabel('Epochs')
ax.set_ylabel('Accuracy Score')
ax.legend(loc='lower right')
ax.set_title('16 Feature Set: Accuracy over iterations for No Hidden Layer')
```




```
NN_1_28 = Sequential()
NN_1_28.add(Dense(28, input_shape = (28,), activation = 'relu'))
NN_1_28.add(Dense(1, activation='sigmoid'))
NN_1_28.summary()
```

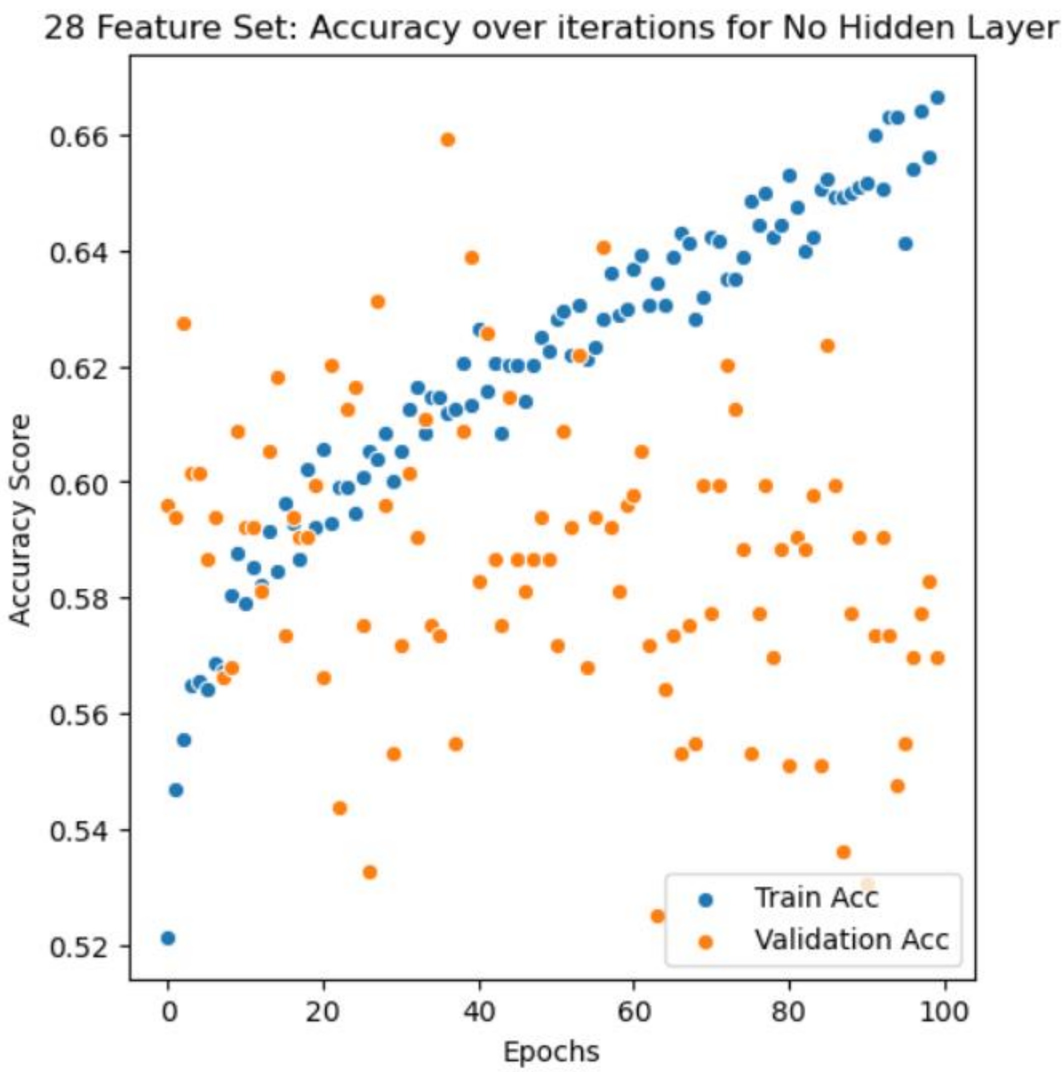
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_13 (Dense)             (None, 28)                812
_________________________________________________________________
dense_14 (Dense)             (None, 1)                 29
=================================================================
Total params: 841
Trainable params: 841
Non-trainable params: 0
_________________________________________________________________
```

```
NN_1_28.compile(Adam(lr = .001), "binary_crossentropy", metrics=["accuracy"])
run_hist_28_1 = NN_1_28.fit(X_28_train_ss, y_16_train, validation_data=(X_28_test_ss, y_28_test), epochs=100)
```

```
n = len(run_hist_28_1.history["val_acc"])
fig = plt.figure(figsize=(12, 6))

ax = fig.add_subplot(1, 2, 2)
sns.scatterplot(x=range(n), y=(run_hist_28_1.history["acc"]), ax=ax, label="Train Acc")
sns.scatterplot(x=range(n), y=(run_hist_28_1.history["val_acc"]), ax=ax, label="Validation Acc")
ax.set_xlabel('Epochs')
ax.set_ylabel('Accuracy Score')
ax.legend(loc='lower right')
ax.set_title('28 Feature Set: Accuracy over iterations for No Hidden Layer')
```

```
Text(0.5, 1.0, '28 Feature Set: Accuracy over iterations for No Hidden Layer')
```



When comparing the results of the 0 Hidden Layer Neural Network, it is striking that for the 16-feature dataset, the training data's accuracy increases as more iterations occur. Initially, there is a sharp increase in the accuracy but subsequently there is an increasing trend in the accuracy but at a smaller rate. However, for the testing or validation data, the accuracy increases first and then more or less remains constant. The accuracy score for testing data is slightly more than the baseline model, seeming to plateau at around 0.86 compared to baseline model's 0.857.

For the 28-feature dataset, the results are diametrically different. While there is an increasing trend in the accuracy of the training data, the accuracy score for the testing data appears to be randomly dispersed with no discernible pattern. It appears that the 28-feature dataset is too much for this simple neural network to handle therefore, its performance is so poor.

1-Hidden Layer Neural Network:

Next, a Neural Network with 1 Hidden Layer was defined. This Neural Network also used a relu (Rectified Linear Unit) activation function for the input. The input layer had 28 filters. The hidden layer, directly after the input layer, had a relu activation function and 28 filters (nodes) as well. Once more, sigmoid activation function was used for the output. A summary of the resulting models was displayed.

For the 16-Feature set, the Neural Network had 1317 trainable parameters. For the 28-Feature set, the Neural Network had a total of 1653 trainable parameters. Neural Networks for both datasets were compiled using the ADAM optimizer. Binary cross entropy was used to estimate their accuracy.

The models for both datasets were run for 100 epochs again and the accuracy for training data and testing data for each epoch was plotted on graphs for better visualization.

```
NN_2_16 = Sequential()
NN_2_16.add(Dense(28, input_shape=(16,), activation='relu'))
NN_2_16.add(Dense(28, activation='relu'))
NN_2_16.add(Dense(1, activation='sigmoid'))
NN_2_16.summary()
```

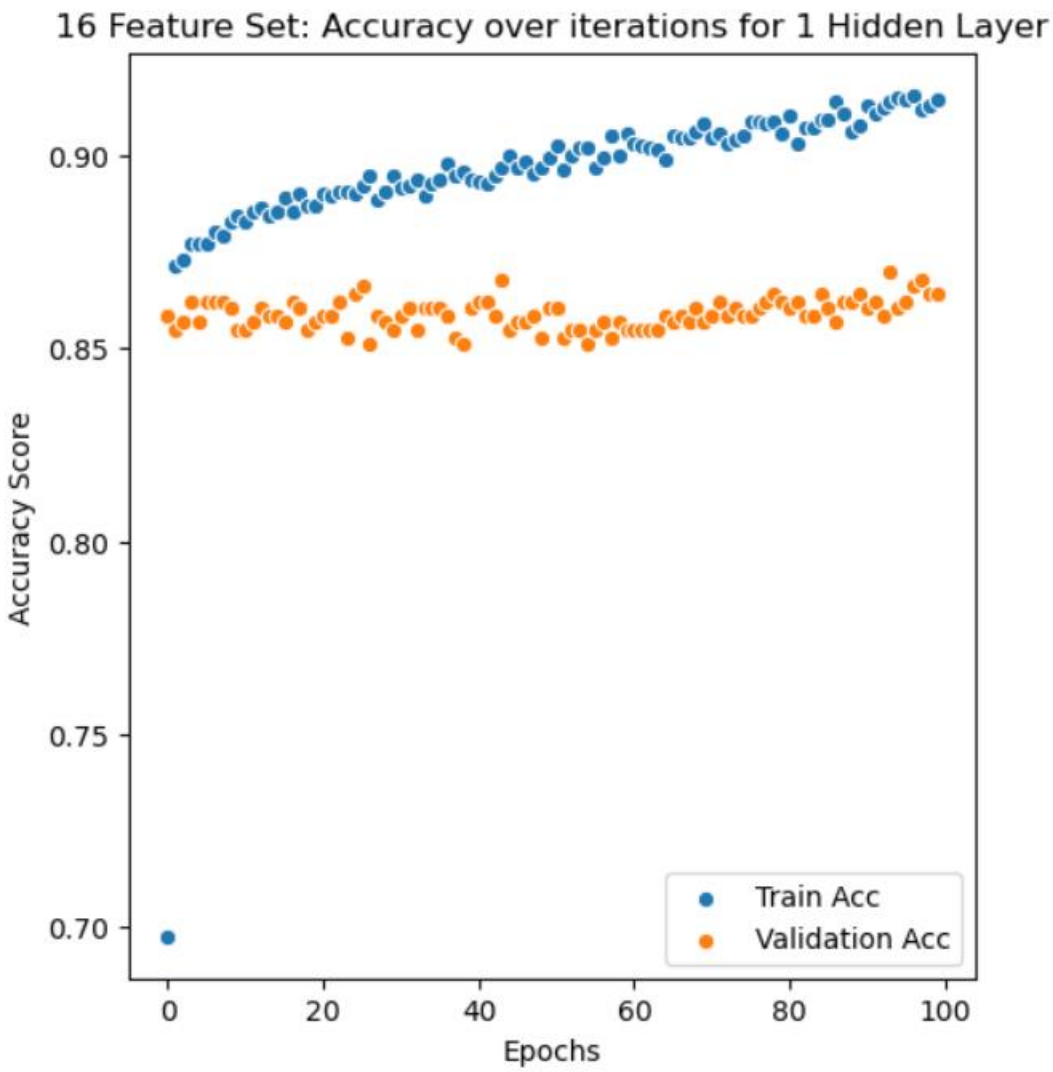
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 28)                476
_________________________________________________________________
dense_16 (Dense)             (None, 28)                812
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 29
=================================================================
Total params: 1,317
Trainable params: 1,317
Non-trainable params: 0
_________________________________________________________________
```

```
NN_2_16.compile(Adam(lr = .001), "binary_crossentropy", metrics=["accuracy"])
run_hist_16_2 = NN_2_16.fit(X_16_train_ss, y_16_train, validation_data=(X_16_test_ss, y_16_test), epochs=100)
```

```
n = len(run_hist_16_2.history["val_acc"])
fig = plt.figure(figsize=(12, 6))

ax = fig.add_subplot(1, 2, 2)
sns.scatterplot(x=range(n), y=(run_hist_16_2.history["acc"]), ax=ax, label="Train Acc")
sns.scatterplot(x=range(n), y=(run_hist_16_2.history["val_acc"]), ax=ax, label="Validation Acc")
ax.set_xlabel('Epochs')
ax.set_ylabel('Accuracy Score')
ax.legend(loc='lower right')
ax.set_title('16 Feature Set: Accuracy over iterations for 1 Hidden Layer')
```

```
Text(0.5, 1.0, '16 Feature Set: Accuracy over iterations for 1 Hidden Layer')
```



```
NN_2_28 = Sequential()
NN_2_28.add(Dense(28, input_shape=(28,), activation='relu'))
NN_2_28.add(Dense(28, activation='relu'))
NN_2_28.add(Dense(1, activation='sigmoid'))
NN_2_28.summary()
```

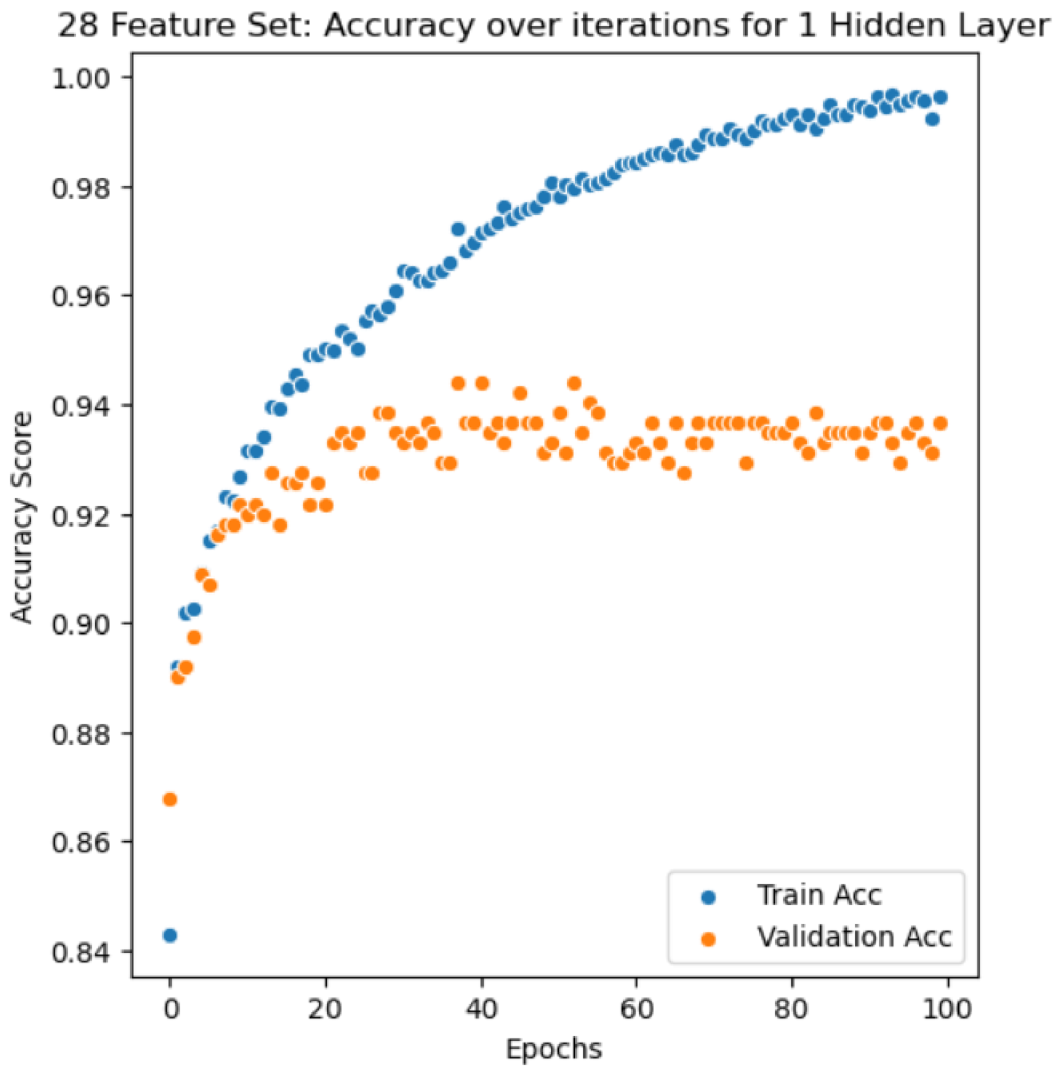
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 28)                812
_________________________________________________________________
dense_19 (Dense)             (None, 28)                812
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 29
=================================================================
Total params: 1,653
Trainable params: 1,653
Non-trainable params: 0
_________________________________________________________________
```

```
NN_2_28.compile(Adam(lr = .001), "binary_crossentropy", metrics=["accuracy"])
run_hist_28_2 = NN_2_28.fit(X_28_train_ss, y_28_train, validation_data=(X_28_test_ss, y_28_test), epochs=100)
```

```
n = len(run_hist_28_2.history["val_acc"])
fig = plt.figure(figsize=(12, 6))

ax = fig.add_subplot(1, 2, 2)
sns.scatterplot(x=range(n), y=(run_hist_28_2.history["acc"]), ax=ax, label="Train Acc")
sns.scatterplot(x=range(n), y=(run_hist_28_2.history["val_acc"]), ax=ax, label="Validation Acc")
ax.set_xlabel('Epochs')
ax.set_ylabel('Accuracy Score')
ax.legend(loc='lower right')
ax.set_title('28 Feature Set: Accuracy over iterations for 1 Hidden Layer')
```

```
Text(0.5, 1.0, '28 Feature Set: Accuracy over iterations for 1 Hidden Layer')
```



For the 1-Hidden Layer Neural Network, some of the results are different. As far as the 16-feature dataset is concerned, there is no perceptible difference. Again, the training accuracy is on an upward trajectory while testing accuracy increases initially and then plateaus at around 0.86.

However, results are different for the 28-feature dataset. There is an upward trend to training accuracy as before. However, this time there is a discernible pattern to testing data's accuracy too. It rises initially and then subsequently plateaus at around 0.935. This appears promising as this is better than the baseline model which had an accuracy of 0.892. It appears that the additional hidden layer was needed for the Neural Network to process the data in the 28-feature dataset.

2-Hidden Layers Neural Network:

Afterwards, the number of Hidden Layers was increased and a Neural Network with 2 Hidden Layers was defined. This Neural Network used a relu (Rectified Linear Unit) activation function for the input layer with 28 filters also. The first hidden layer after the input layer used a relu activation function with 28 filters. The second hidden layer after the first hidden layer also used a relu activation function and had 28 filters also. Finally, the sigmoid activation function was used for the output as it is very accurate for classification. A summary of the resulting models was displayed.

For the 16-Feature set, the Neural Network had 2129 trainable parameters. For the 28-Feature set, the Neural Network had a total of 2465 trainable parameters. Neural Networks for both datasets were compiled using the ADAM optimizer. Binary cross entropy was used to estimate their accuracy.

The models for both datasets were run for 100 epochs which is the standard throughout the project. The accuracy for training data and testing data for each epoch was plotted on graphs for better visualization.

```
NN_3_16 = Sequential()
NN_3_16.add(Dense(28, input_shape=(16,), activation='relu'))
NN_3_16.add(Dense(28, activation='relu'))
NN_3_16.add(Dense(28, activation='relu'))
NN_3_16.add(Dense(1, activation='sigmoid'))
NN_3_16.summary()
```

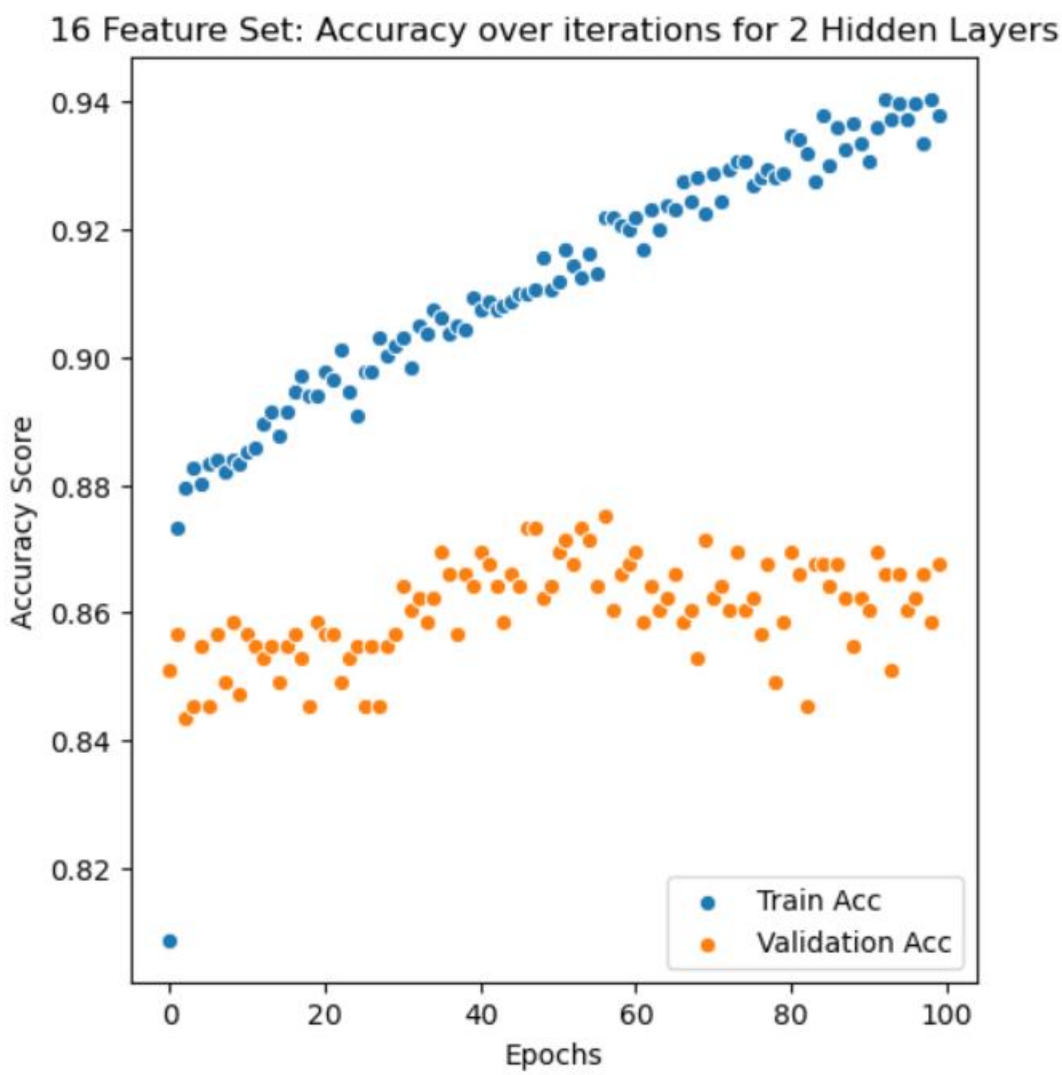
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_21 (Dense)             (None, 28)                476
_________________________________________________________________
dense_22 (Dense)             (None, 28)                812
_________________________________________________________________
dense_23 (Dense)             (None, 28)                812
_________________________________________________________________
dense_24 (Dense)             (None, 1)                 29
=================================================================
Total params: 2,129
Trainable params: 2,129
Non-trainable params: 0
_________________________________________________________________
```

```
NN_3_16.compile(Adam(lr = .001), "binary_crossentropy", metrics=["accuracy"])
run_hist_16_3 = NN_3_16.fit(X_16_train_ss, y_16_train, validation_data=(X_16_test_ss, y_16_test), epochs=100)
```

```
n = len(run_hist_16_3.history["val_acc"])
fig = plt.figure(figsize=(12, 6))

ax = fig.add_subplot(1, 2, 2)
sns.scatterplot(x=range(n), y=(run_hist_16_3.history["acc"]), ax=ax, label="Train Acc")
sns.scatterplot(x=range(n), y=(run_hist_16_3.history["val_acc"]), ax=ax, label="Validation Acc")
ax.set_xlabel('Epochs')
ax.set_ylabel('Accuracy Score')
ax.legend(loc='lower right')
ax.set_title('16 Feature Set: Accuracy over iterations for 2 Hidden Layers')
```

```
Text(0.5, 1.0, '16 Feature Set: Accuracy over iterations for 2 Hidden Layers')
```



```
NN_3_28 = Sequential()
NN_3_28.add(Dense(28, input_shape=(28,), activation='relu'))
NN_3_28.add(Dense(28, activation='relu'))
NN_3_28.add(Dense(28, activation='relu'))
NN_3_28.add(Dense(1, activation='sigmoid'))
NN_3_28.summary()
```

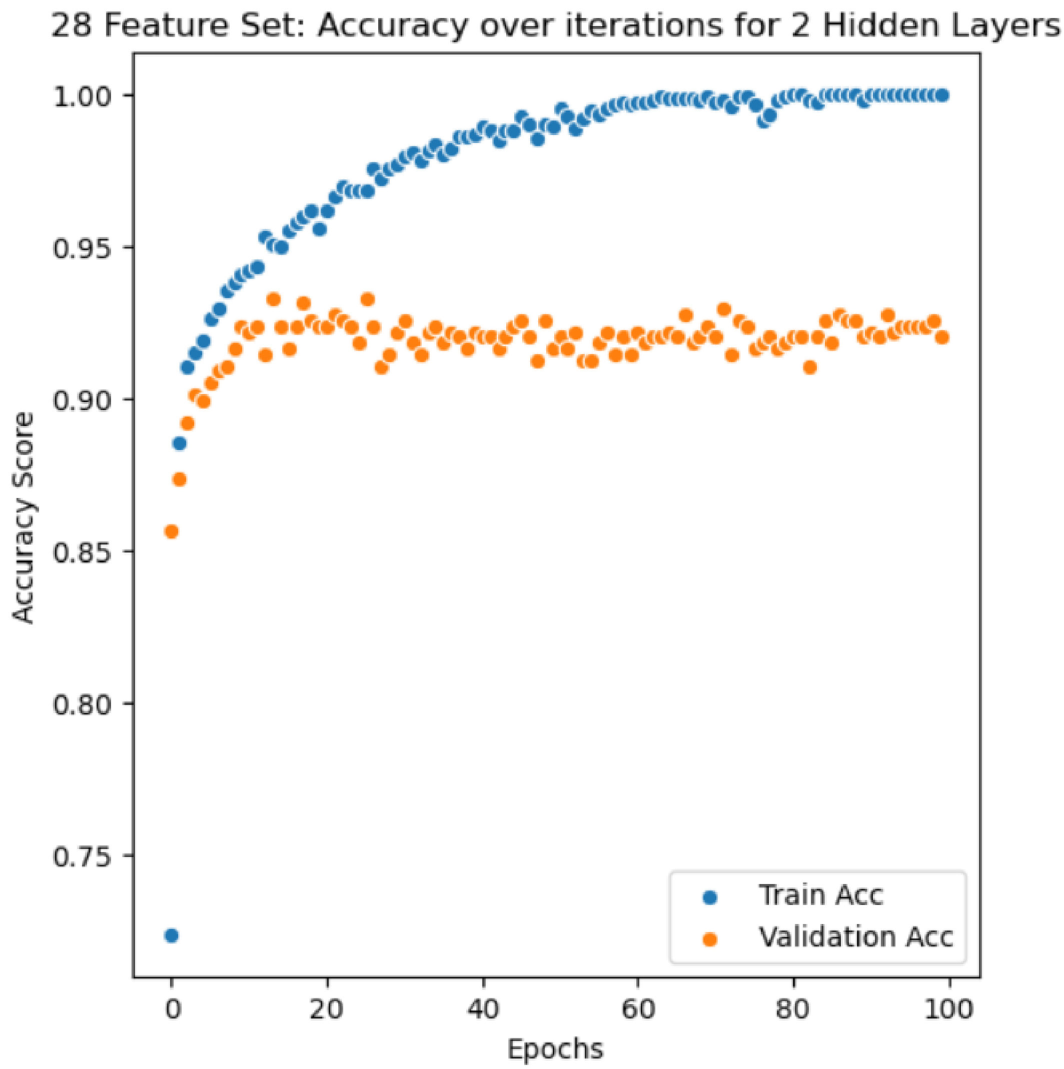
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 28)                812
_________________________________________________________________
dense_26 (Dense)             (None, 28)                812
_________________________________________________________________
dense_27 (Dense)             (None, 28)                812
_________________________________________________________________
dense_28 (Dense)             (None, 1)                 29
=================================================================
Total params: 2,465
Trainable params: 2,465
Non-trainable params: 0
_________________________________________________________________
```

```
NN_3_28.compile(Adam(lr = .001), "binary_crossentropy", metrics=["accuracy"])
run_hist_28_3 = NN_3_28.fit(X_28_train_ss, y_28_train, validation_data=(X_28_test_ss, y_28_test), epochs=100)
```

```
n = len(run_hist_28_3.history["val_acc"])
fig = plt.figure(figsize=(12, 6))

ax = fig.add_subplot(1, 2, 2)
sns.scatterplot(x=range(n), y=(run_hist_28_3.history["acc"]), ax=ax, label="Train Acc")
sns.scatterplot(x=range(n), y=(run_hist_28_3.history["val_acc"]), ax=ax, label="Validation Acc")
ax.set_xlabel('Epochs')
ax.set_ylabel('Accuracy Score')
ax.legend(loc='lower right')
ax.set_title('28 Feature Set: Accuracy over iterations for 2 Hidden Layers')
```

```
Text(0.5, 1.0, '28 Feature Set: Accuracy over iterations for 2 Hidden Layers')
```



The results of the 2-Hidden Layers Network are not particularly surprising. For the 16-feature dataset, the results are not particularly different from before. The training accuracy keeps increasing with iterations while the testing data's accuracy increases initially and then appears to revolve around the 0.86 mark noted before. There seems to be more variation in the results however, compared to the previous 1-Layer Neural Network for the testing data.

For the 28-feature dataset, the results are the same as before. The training accuracy shows an upward trend while testing accuracy rises and then plateaus at around 0.92 accuracy. This appears to be very slightly less than before.

3-Hidden Layers Neural Network:

Finally, the number of Hidden Layers was increased again and a Neural Network with 3 Hidden Layers was created. This Neural Network used a relu (Rectified Linear Unit) activation function for the input layer with 28 filters and the first hidden layer after the input layer used a relu activation function with 28 filters. The second hidden layer after the first hidden layer also used a relu activation function and had 28 filters and the third hidden layer also used a relu activation function with 28 filters. Finally, the sigmoid activation function was used for the output. A summary of the resulting models was displayed.

For the 16-Feature set, the Neural Network had 2941 trainable parameters. For the 28-Feature set, the Neural Network had a total of 3277 trainable parameters. Neural Networks for both datasets were compiled using the ADAM optimizer. Binary cross entropy was used to estimate their accuracy.

The models for both datasets were run for the standard 100 epochs and the accuracy for training data and testing data for each epoch was plotted on graphs for better visualization.

```
NN_4_16 = Sequential()
NN_4_16.add(Dense(28, input_shape=(16,), activation='relu'))
NN_4_16.add(Dense(28, activation='relu'))
NN_4_16.add(Dense(28, activation='relu'))
NN_4_16.add(Dense(28, activation='relu'))
NN_4_16.add(Dense(1, activation='sigmoid'))
NN_4_16.summary()
```

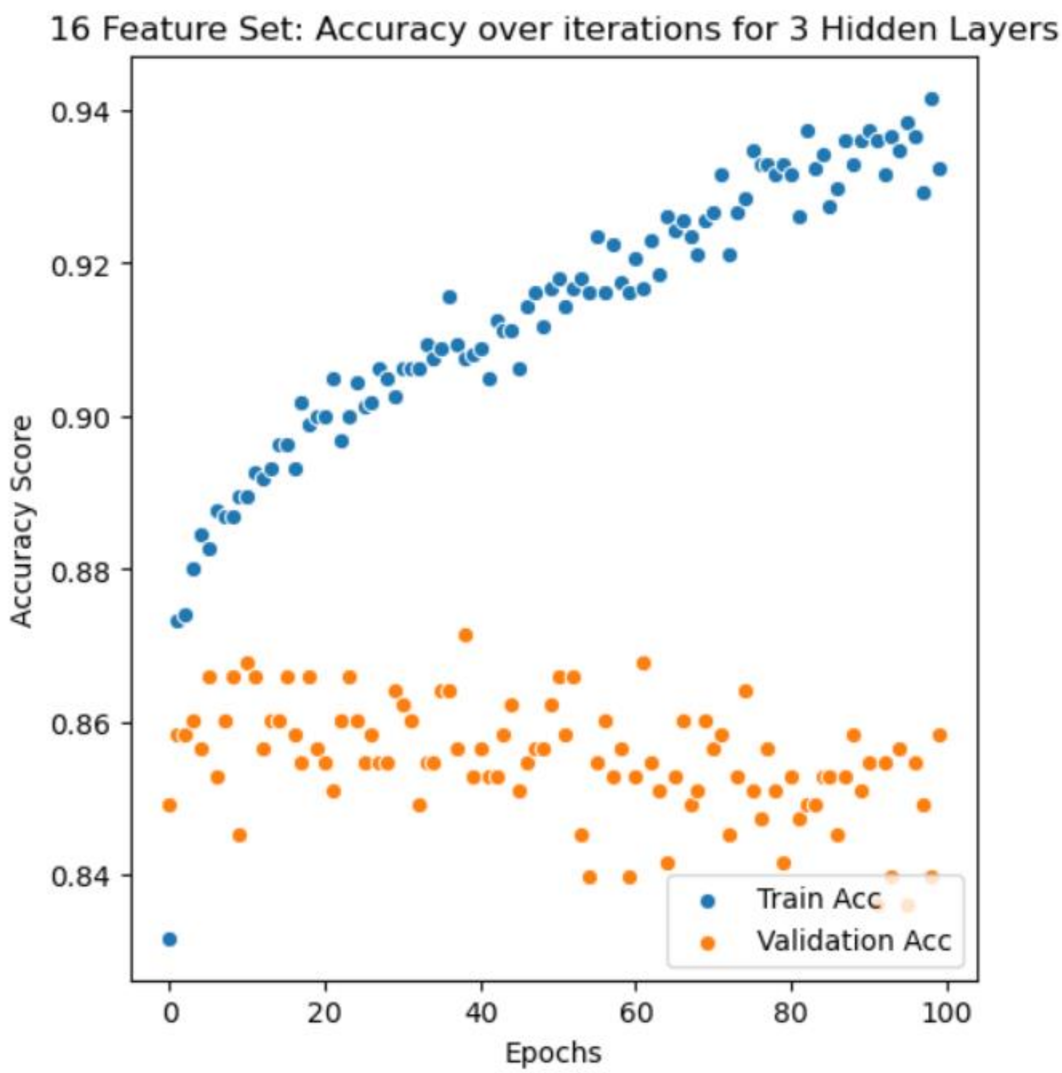
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_29 (Dense)             (None, 28)                476
_________________________________________________________________
dense_30 (Dense)             (None, 28)                812
_________________________________________________________________
dense_31 (Dense)             (None, 28)                812
_________________________________________________________________
dense_32 (Dense)             (None, 28)                812
_________________________________________________________________
dense_33 (Dense)             (None, 1)                 29
=================================================================
Total params: 2,941
Trainable params: 2,941
Non-trainable params: 0
_________________________________________________________________
```

```
NN_4_16.compile(Adam(lr = .001), "binary_crossentropy", metrics=["accuracy"])
run_hist_16_4 = NN_4_16.fit(X_16_train_ss, y_16_train, validation_data=(X_16_test_ss, y_16_test), epochs=100)
```

```
n = len(run_hist_16_4.history["val_acc"])
fig = plt.figure(figsize=(12, 6))

ax = fig.add_subplot(1, 2, 2)
sns.scatterplot(x=range(n), y=(run_hist_16_4.history["acc"]), ax=ax, label="Train Acc")
sns.scatterplot(x=range(n), y=(run_hist_16_4.history["val_acc"]), ax=ax, label="Validation Acc")
ax.set_xlabel('Epochs')
ax.set_ylabel('Accuracy Score')
ax.legend(loc='lower right')
ax.set_title('16 Feature Set: Accuracy over iterations for 3 Hidden Layers')
```

```
Text(0.5, 1.0, '16 Feature Set: Accuracy over iterations for 3 Hidden Layers')
```



```
NN_4_28 = Sequential()
NN_4_28.add(Dense(28, input_shape=(28,), activation='relu'))
NN_4_28.add(Dense(28, activation='relu'))
NN_4_28.add(Dense(28, activation='relu'))
NN_4_28.add(Dense(28, activation='relu'))
NN_4_28.add(Dense(1, activation='sigmoid'))
NN_4_28.summary()
```

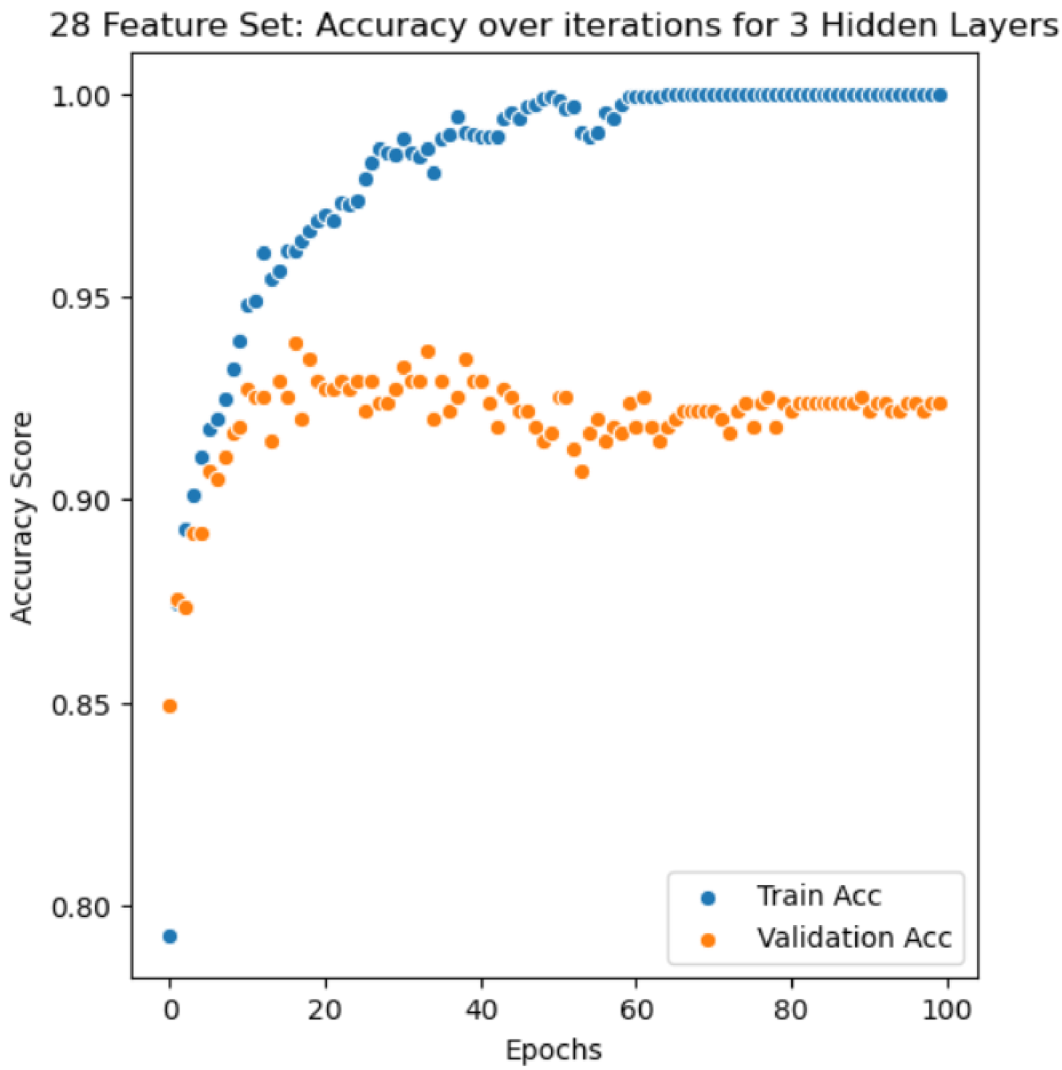
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_34 (Dense)             (None, 28)                812
_________________________________________________________________
dense_35 (Dense)             (None, 28)                812
_________________________________________________________________
dense_36 (Dense)             (None, 28)                812
_________________________________________________________________
dense_37 (Dense)             (None, 28)                812
_________________________________________________________________
dense_38 (Dense)             (None, 1)                 29
=================================================================
Total params: 3,277
Trainable params: 3,277
Non-trainable params: 0
_________________________________________________________________
```

```
NN_4_28.compile(Adam(lr = .001), "binary_crossentropy", metrics=["accuracy"])
run_hist_28_4 = NN_4_28.fit(X_28_train_ss, y_28_train, validation_data=(X_28_test_ss, y_28_test), epochs=100)
```

```
n = len(run_hist_28_4.history["val_acc"])
fig = plt.figure(figsize=(12, 6))

ax = fig.add_subplot(1, 2, 2)
sns.scatterplot(x=range(n), y=(run_hist_28_4.history["acc"]), ax=ax, label="Train Acc")
sns.scatterplot(x=range(n), y=(run_hist_28_4.history["val_acc"]), ax=ax, label="Validation Acc")
ax.set_xlabel('Epochs')
ax.set_ylabel('Accuracy Score')
ax.legend(loc='lower right')
ax.set_title('28 Feature Set: Accuracy over iterations for 3 Hidden Layers')
```

```
Text(0.5, 1.0, '28 Feature Set: Accuracy over iterations for 3 Hidden Layers')
```



The results for the 3-Hidden Layer Neural Network are slightly worse than before. For the 16-feature dataset, the training accuracy keeps increasing with iterations as before but for testing accuracy, there is a lot of variation in the accuracy values. The values appear to be centered around 0.86 but that is not particularly clear.

For the 28-feature set, the results are almost similar. The training accuracy keeps rising. The testing accuracy, as before initially rises. Then it seems to oscillate, then dips a little until plateauing at around 0.92.

Results:

The results of all the models were summed up by calculating the mean across all iterations. The following table was constructed to present them in a lucid fashion:

	16 Features Dataset	28 Features Dataset
Random Forest	0.856611	0.891993
No Hidden Layer	0.858827	0.586834
One Hidden Layer	0.858845	0.930242
Two Hidden Layers	0.860410	0.918715
Three Hidden Layers	0.855196	0.920372

The classification results indicate, the baseline model did a decent job with both datasets. The subsequent introduction of neural networks seems to have had no impact on the 16-feature dataset whose results remain relatively unchanged even as the number of hidden layers were increased. The 0-Hidden Layer Network model seems to have produced a very poor result with the 28-feature dataset. This may be because, as observed previously, the model was unable to handle the data well. However, introduction of the 1st hidden layer improved the results of the 28-feature dataset beyond that of the baseline model. Subsequently, additional layers did not improve the accuracy score of the 28-feature dataset.

From the results presented above, 1-Hidden Layer Neural Network seems to be the best choice for this dataset with the given conditions. Admittedly though, additional layers did not have a significant impact on the results.

Key Findings:

The key findings from this analysis are as follows:

1. Neural Networks with hidden layers can provide comparable if not better performance than regular classification techniques.
2. The performance of neural networks improves as more data is made available to them.
3. Neural Networks perform better with hidden layers. Without hidden layers, when provided with lots of data, they perform worse than regular classifiers.
4. The performance of neural networks does not increase monotonically with more hidden layers. After a limit, which is dependent on the amount of data, additional hidden layers do not improve performance.

Issues:

Some issues with the project setup and the techniques used within it are taken note of here such as:

1. The datasets used in the analysis are not particularly large. With larger datasets, the project may yield more incisive results as neural networks work better with large amounts of data.
2. The neural nets used in the project do not incorporate any regularization techniques. Adding these techniques may yield different results.
3. The models used in the project were run only once for 100 epochs each. If these models were run for 100 epochs for multiple iterations and the results averaged, then the results produced would be more robust.

Future Steps:

Some future steps for dealing with issues outlined above are listed below:

1. Larger datasets may be used for this project to give more incisive results.
2. Regularization techniques may be incorporated in the models used in the project for improved results.
3. The models could be run for multiple iterations for greater than 100 epochs each time to give more robust results.
4. Various parameters such as activation functions and filters at each layer may be tuned to produce more results.

The steps outlined above, if implemented separately, may improve project conclusions on their own. However, these steps can be merged in any combination to produce results that may yield additional insights about deep learning with neural networks.