Practical No 02 (A)

Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition dataset Data set Link :- https://archive.ics.uci.edu/ml/datasets/letter+recognition

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

from keras.models import Sequential
from keras.layers import Dense
from keras import callbacks
```

```
df = pd.read_csv('letter-recognition.data', header=None)
```

```
df.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
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

```
df.columns = ['letter', 'x-box', 'y-box', 'width', 'high', 'onpix', 'x-bar', 'y-bar', 'x2bar', 'y2bar', 'xy
```

```
df
```

	letter	x-box	y-box	width	high	onpix	x-bar	y-bar	x2bar	y2bar	xybar	x2
0	T	2	8	3	5	1	8	13	0	6	6	
1	I	5	12	3	7	2	10	5	5	4	13	
2	D	4	11	6	8	6	10	6	2	6	10	
3	N	7	11	6	6	3	5	9	4	6	4	
4	G	2	1	3	1	1	8	6	6	6	6	
...	...	...	...	...	...	...	...	...	...	...	...	
19995	D	2	2	3	3	2	7	7	7	6	6	
19996	C	7	10	8	8	4	4	8	6	9	12	
19997	T	6	9	6	7	5	6	11	3	7	11	
19998	S	2	3	4	2	1	8	7	2	6	10	
19999	A	4	9	6	6	2	9	5	3	1	8	

20000 rows × 17 columns

```
df['letter'].value_counts(normalize=True).mul(100).round(1).astype(str) + '%'
```

```
U    4.1%
D    4.0%
P    4.0%
T    4.0%
M    4.0%
A    3.9%
X    3.9%
Y    3.9%
N    3.9%
Q    3.9%
F    3.9%
G    3.9%
E    3.8%
B    3.8%
V    3.8%
L    3.8%
R    3.8%
I    3.8%
O    3.8%
W    3.8%
S    3.7%
J    3.7%
K    3.7%
C    3.7%
H    3.7%
Z    3.7%
Name: letter, dtype: object
```

```
df_without_letter_column = df.loc[:, df.columns != 'letter']
letter_column = df['letter']
x = df_without_letter_column.values
min_max_scaler = MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df = pd.DataFrame(x_scaled)
df.insert(0, 'letter', letter_column)
```

```
df.columns = ['letter', 'x-box', 'y-box', 'width', 'high', 'onpix', 'x-bar', 'y-bar', 'x2bar', 'y2bar', 'xy
df
```

	letter	x-box	y-box	width	high	onpix	x-bar	y-bar
0	T	0.133333	0.533333	0.200000	0.333333	0.066667	0.533333	0.866667
1	I	0.333333	0.800000	0.200000	0.466667	0.133333	0.666667	0.333333
2	D	0.266667	0.733333	0.400000	0.533333	0.400000	0.666667	0.400000
3	N	0.466667	0.733333	0.400000	0.400000	0.200000	0.333333	0.600000
4	G	0.133333	0.066667	0.200000	0.066667	0.066667	0.533333	0.400000
...	...	...	...	...	...	...	...	...
19995	D	0.133333	0.133333	0.200000	0.200000	0.133333	0.466667	0.466667
19996	C	0.466667	0.666667	0.533333	0.533333	0.266667	0.266667	0.533333
19997	T	0.400000	0.600000	0.400000	0.466667	0.333333	0.400000	0.733333
19998	S	0.133333	0.200000	0.266667	0.133333	0.066667	0.533333	0.466667
19999	A	0.266667	0.600000	0.400000	0.400000	0.133333	0.600000	0.333333

20000 rows × 17 columns

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

```
X_train = X.loc[:15999]
y_train = y.loc[:15999]
X_test = X.loc[16000:]
y_test = y.loc[16000:]
```

```
def evaluate(model):
    model.fit(X_train, y_train)
    x_train_prediction = model.predict(X_train)
    x_test_prediction = model.predict(X_test)
    train_accuracy = accuracy_score(y_train, x_train_prediction)
    test_accuracy = accuracy_score(y_test, x_test_prediction)
    clf_report = classification_report(y_test, x_test_prediction)
    print(f'Train set accuracy: {train_accuracy:.2f}')
    print(f'Test set accuracy: {test_accuracy:.2f}' + '\n')
    print(clf_report + '\n')
```

```
model = KNeighborsClassifier()
evaluate(model)
```

```
Train set accuracy: 0.98
Test set accuracy: 0.95

              precision    recall  f1-score   support

           A       0.98      0.99      0.99       156
           B       0.86      0.97      0.91       136
           C       0.96      0.96      0.96       142
           D       0.89      0.98      0.93       167
           E       0.90      0.94      0.92       152
           F       0.91      0.93      0.92       153
           G       0.97      0.93      0.95       164
           H       0.84      0.87      0.86       151
           I       0.93      0.96      0.95       165
           J       0.96      0.92      0.94       148
           K       0.92      0.89      0.91       146
           L       0.99      0.97      0.98       157
           M       0.97      0.97      0.97       144
           N       0.96      0.90      0.93       166
           O       0.94      0.94      0.94       139
           P       0.96      0.89      0.93       168
           Q       0.94      0.94      0.94       168
           R       0.92      0.93      0.92       161
           S       0.98      0.95      0.97       161
           T       1.00      0.95      0.98       151
           U       0.99      0.99      0.99       168
           V       0.94      0.98      0.96       136
           W       0.99      0.98      0.99       139
           X       0.95      0.94      0.94       159
           Y       0.97      0.96      0.97       145
           Z       0.97      0.97      0.97       158

    accuracy                           0.95      4000
   macro avg       0.95      0.95      0.95      4000
weighted avg       0.95      0.95      0.95      4000
```

```
k_range = list(range(1, 9))
param_grid = dict(n_neighbors=k_range)

grid = GridSearchCV(model, param_grid, cv=10, scoring='accuracy', return_train_score=False, verbose=2)

grid_search=grid.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 8 candidates, totalling 80 fits
[CV] END .....................................n_neighbors=1; total time=   0.2s
[CV] END .....................................n_neighbors=1; total time=   0.2s
[CV] END .....................................n_neighbors=1; total time=   0.2s
[CV] END .....................................n_neighbors=1; total time=   0.2s
[CV] END .....................................n_neighbors=1; total time=   0.2s
[CV] END .....................................n_neighbors=1; total time=   0.2s
[CV] END .....................................n_neighbors=1; total time=   0.3s
[CV] END .....................................n_neighbors=1; total time=   0.7s
[CV] END .....................................n_neighbors=1; total time=   0.7s
[CV] END .....................................n_neighbors=1; total time=   0.4s
```

```
[CV] END ....................................n_neighbors=2; total time=   0.5s
[CV] END ....................................n_neighbors=2; total time=   0.2s
[CV] END ....................................n_neighbors=2; total time=   0.2s
[CV] END ....................................n_neighbors=2; total time=   0.2s
[CV] END ....................................n_neighbors=2; total time=   0.2s
[CV] END ....................................n_neighbors=2; total time=   0.2s
[CV] END ....................................n_neighbors=2; total time=   0.2s
[CV] END ....................................n_neighbors=2; total time=   0.2s
[CV] END ....................................n_neighbors=2; total time=   0.2s
[CV] END ....................................n_neighbors=2; total time=   0.2s
[CV] END ....................................n_neighbors=3; total time=   0.2s
[CV] END ....................................n_neighbors=3; total time=   0.2s
[CV] END ....................................n_neighbors=3; total time=   0.2s
[CV] END ....................................n_neighbors=3; total time=   0.2s
[CV] END ....................................n_neighbors=3; total time=   0.2s
[CV] END ....................................n_neighbors=3; total time=   0.2s
[CV] END ....................................n_neighbors=3; total time=   0.2s
[CV] END ....................................n_neighbors=3; total time=   0.2s
[CV] END ....................................n_neighbors=3; total time=   0.2s
[CV] END ....................................n_neighbors=3; total time=   0.2s
[CV] END ....................................n_neighbors=4; total time=   0.2s
[CV] END ....................................n_neighbors=4; total time=   0.2s
[CV] END ....................................n_neighbors=4; total time=   0.2s
[CV] END ....................................n_neighbors=4; total time=   0.2s
[CV] END ....................................n_neighbors=4; total time=   0.2s
[CV] END ....................................n_neighbors=4; total time=   0.2s
[CV] END ....................................n_neighbors=4; total time=   0.2s
[CV] END ....................................n_neighbors=4; total time=   0.2s
[CV] END ....................................n_neighbors=4; total time=   0.2s
[CV] END ....................................n_neighbors=4; total time=   0.2s
[CV] END ....................................n_neighbors=5; total time=   0.2s
[CV] END ....................................n_neighbors=5; total time=   0.2s
[CV] END ....................................n_neighbors=5; total time=   0.2s
[CV] END ....................................n_neighbors=5; total time=   0.2s
[CV] END ....................................n_neighbors=5; total time=   0.2s
[CV] END ....................................n_neighbors=5; total time=   0.2s
[CV] END ....................................n_neighbors=5; total time=   0.2s
[CV] END ....................................n_neighbors=5; total time=   0.2s
[CV] END ....................................n_neighbors=5; total time=   0.2s
[CV] END ....................................n_neighbors=5; total time=   0.2s
[CV] END ....................................n_neighbors=6; total time=   0.2s
[CV] END ....................................n_neighbors=6; total time=   0.2s
[CV] END ....................................n_neighbors=6; total time=   0.2s
[CV] END ....................................n_neighbors=6; total time=   0.2s
[CV] END ....................................n_neighbors=6; total time=   0.2s
[CV] END ....................................n_neighbors=6; total time=   0.2s
[CV] END ....................................n_neighbors=6; total time=   0.2s
```

```
print(grid_search.best_params_)
```

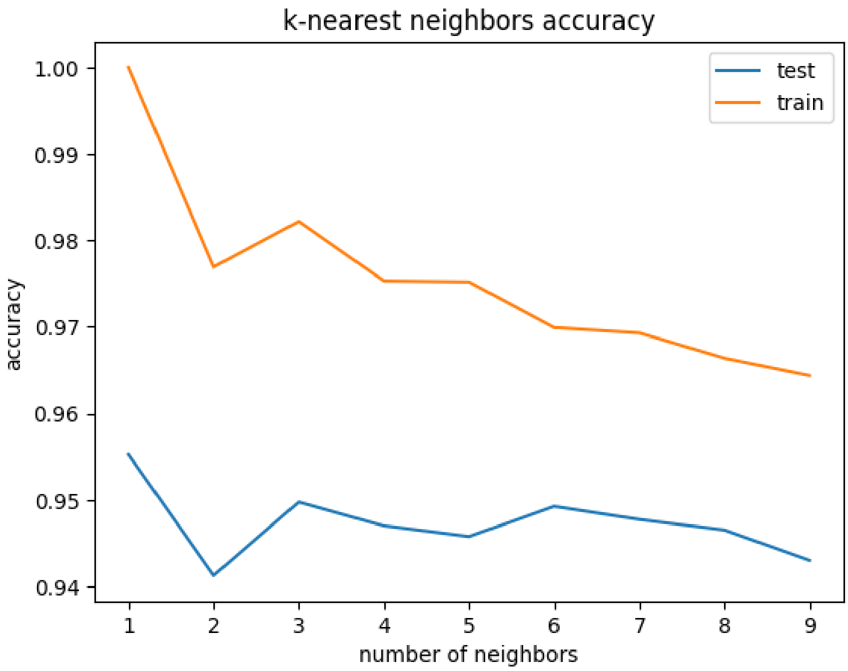
```
{'n_neighbors': 1}
```

```
model = KNeighborsClassifier(**grid_search.best_params_)
evaluate(model)
```

```
Train set accuracy: 1.00
Test set accuracy: 0.96

              precision    recall  f1-score   support

           A       0.99      0.99      0.99       156
           B       0.91      0.95      0.93       136
           C       0.96      0.96      0.96       142
           D       0.94      0.97      0.95       167
           E       0.96      0.92      0.94       152
           F       0.92      0.92      0.92       153
           G       0.96      0.96      0.96       164
           H       0.91      0.89      0.90       151
           I       0.95      0.97      0.96       165
           J       0.97      0.94      0.96       148
           K       0.91      0.91      0.91       146
           L       0.97      0.97      0.97       157
           M       0.99      0.97      0.98       144
           N       0.97      0.95      0.96       166
           O       0.94      0.97      0.95       139
           P       0.95      0.93      0.94       168
           Q       0.95      0.96      0.95       168
           R       0.90      0.91      0.91       161
```

```
           S       0.98      0.98      0.98       161
           T       0.99      0.97      0.98       151
           U       0.98      0.99      0.99       168
           V       0.94      0.99      0.96       136
           W       0.98      0.98      0.98       139
           X       0.97      0.96      0.97       159
           Y       0.99      0.97      0.98       145
           Z       0.96      0.98      0.97       158

    accuracy                           0.96      4000
   macro avg       0.96      0.96      0.96      4000
weighted avg       0.96      0.96      0.96      4000
```

```python
no_neighbors = np.arange(1, 10, 1)
train_accuracy = np.empty(len(no_neighbors))
test_accuracy = np.empty(len(no_neighbors))

for i, k in enumerate(no_neighbors):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train,y_train)
    train_accuracy[i] = model.score(X_train, y_train)
    test_accuracy[i] = model.score(X_test, y_test)

plt.title('k-nearest neighbors accuracy')
plt.plot(no_neighbors, test_accuracy, label='test')
plt.plot(no_neighbors, train_accuracy, label='train')
plt.legend()
plt.xlabel('number of neighbors')
plt.ylabel('accuracy')
plt.show()
```




```python
encoder = LabelEncoder()

encoder.fit(y_train)
train_labels = encoder.transform(y_train)

encoder.fit(y_test)
test_labels = encoder.transform(y_test)
```

```python
model = Sequential()
model.add(Dense(units=64, activation='relu', input_shape=(16,)))
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=26, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy',
```

```
            optimizer='adam',
            metrics=['accuracy'])

early_stopping = callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.001,
    patience=10,
    restore_best_weights=True)

history = model.fit(X_train, train_labels, validation_data=(X_test, test_labels), epochs=200, batch_size=1(
```

```
Epoch 11/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.6958 - accuracy: 0.7999 - val_loss:
Epoch 12/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.6654 - accuracy: 0.8040 - val_loss:
Epoch 13/200
1000/1000 [==============================] - 4s 4ms/step - loss: 0.6387 - accuracy: 0.8113 - val_loss:
Epoch 14/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.6116 - accuracy: 0.8183 - val_loss:
Epoch 15/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.5844 - accuracy: 0.8292 - val_loss:
Epoch 16/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.5652 - accuracy: 0.8313 - val_loss:
Epoch 17/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.5434 - accuracy: 0.8371 - val_loss:
Epoch 18/200
1000/1000 [==============================] - 3s 3ms/step - loss: 0.5229 - accuracy: 0.8432 - val_loss:
Epoch 19/200
1000/1000 [==============================] - 3s 3ms/step - loss: 0.5044 - accuracy: 0.8487 - val_loss:
Epoch 20/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.4914 - accuracy: 0.8535 - val_loss:
Epoch 21/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.4749 - accuracy: 0.8558 - val_loss:
Epoch 22/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.4595 - accuracy: 0.8615 - val_loss:
Epoch 23/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.4463 - accuracy: 0.8652 - val_loss:
Epoch 24/200
1000/1000 [==============================] - 3s 3ms/step - loss: 0.4317 - accuracy: 0.8683 - val_loss:
Epoch 25/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.4159 - accuracy: 0.8738 - val_loss:
Epoch 26/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.4073 - accuracy: 0.8767 - val_loss:
Epoch 27/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.3985 - accuracy: 0.8776 - val_loss:
Epoch 28/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.3834 - accuracy: 0.8827 - val_loss:
Epoch 29/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.3758 - accuracy: 0.8842 - val_loss:
Epoch 30/200
1000/1000 [==============================] - 3s 3ms/step - loss: 0.3642 - accuracy: 0.8892 - val_loss:
Epoch 31/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.3565 - accuracy: 0.8906 - val_loss:
Epoch 32/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.3508 - accuracy: 0.8925 - val_loss:
Epoch 33/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.3390 - accuracy: 0.8954 - val_loss:
Epoch 34/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.3352 - accuracy: 0.8970 - val_loss:
Epoch 35/200
1000/1000 [==============================] - 3s 3ms/step - loss: 0.3271 - accuracy: 0.8991 - val_loss:
Epoch 36/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.3230 - accuracy: 0.8967 - val_loss:
Epoch 37/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.3112 - accuracy: 0.9044 - val_loss:
Epoch 38/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.3051 - accuracy: 0.9040 - val_loss:
Epoch 39/200
1000/1000 [==============================] - 2s 2ms/step - loss: 0.3007 - accuracy: 0.9052 - val_loss:
```

```
x_train_prediction = model.predict(X_train)
y_classes = x_train_prediction.argmax(axis=-1)
train_accuracy = accuracy_score(train_labels, y_classes)
print(f'Train set accuracy: {train_accuracy:.2f}')
```

```
500/500 [==============================] - 1s 1ms/step
Train set accuracy: 0.95
```

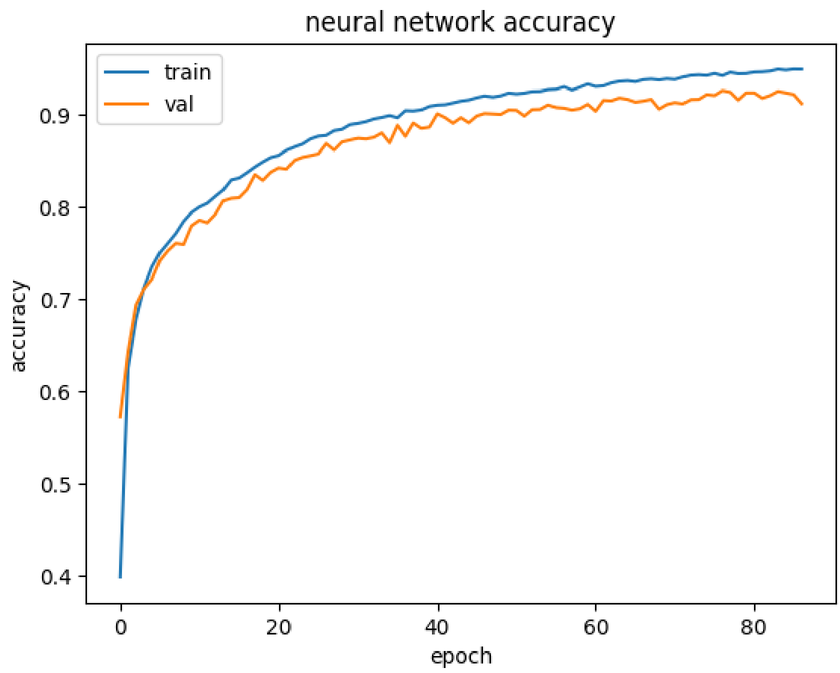
```
x_test_prediction = model.predict(X_test)
y_classes = x_test_prediction.argmax(axis=-1)
test_accuracy = accuracy_score(test_labels, y_classes)
print(f'Test set accuracy: {test_accuracy:.2f}')
```

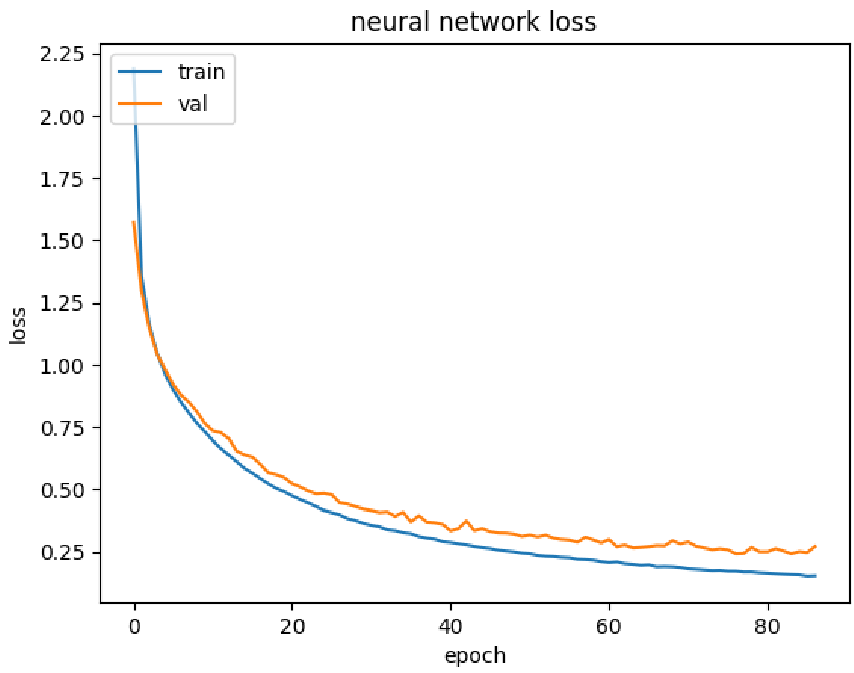
```
125/125 [==============================] - 0s 1ms/step
Test set accuracy: 0.93
```

```
clf_report = classification_report(encoder.inverse_transform(test_labels), encoder.inverse_transform(y_clas
print(clf_report)
```

```
              precision    recall  f1-score   support

           A       0.97      0.95      0.96       156
           B       0.89      0.96      0.92       136
           C       0.94      0.94      0.94       142
           D       0.87      0.95      0.91       167
           E       0.92      0.86      0.89       152
           F       0.89      0.90      0.89       153
           G       0.89      0.87      0.88       164
           H       0.87      0.84      0.86       151
           I       0.97      0.88      0.92       165
           J       0.91      0.97      0.93       148
           K       0.86      0.96      0.91       146
           L       0.94      0.95      0.95       157
           M       0.95      0.98      0.97       144
           N       0.97      0.91      0.94       166
           O       0.88      0.94      0.91       139
           P       0.91      0.94      0.92       168
           Q       0.95      0.92      0.93       168
           R       0.88      0.88      0.88       161
           S       0.96      0.88      0.92       161
           T       0.90      0.97      0.94       151
           U       0.98      0.97      0.97       168
           V       0.98      0.93      0.95       136
           W       0.99      0.99      0.99       139
           X       0.93      0.88      0.91       159
           Y       0.93      0.92      0.93       145
           Z       0.98      0.96      0.97       158

    accuracy                           0.93      4000
   macro avg       0.93      0.93      0.93      4000
weighted avg       0.93      0.93      0.93      4000
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('neural network accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('neural network loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```





✓ 1s completed at 1:21 PM