Assignment No 2A: Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition dataset https://archive.ics.uci.edu/ml/datasets/letter+recognition

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

```
columns = ["lettr", "x-box", "y-box", "width", "height", "onpix", "x-bar","y-bar", "x2bar", "y2bar", "xybar", "x2ybr", "xy2br", "x-ege", "xegvy", "y-ege", "yeg
```

```
df = pd.read_csv('2_letter_recognition.data', names=columns)
```

```
df.shape
```

```
(20000, 17)
```

```
df.head
```

```
<bound method NDFrame.head of       lettr  x-box  y-box  width  height  onpix  x-bar  y-bar  x2bar  y2bar  \
0         T      2      8      3       5      1      8     13      0      6
1         I      5     12      3       7      2     10      5      5      4
2         D      4     11      6       8      6     10      6      2      6
3         N      7     11      6       6      3      5      9      4      6
4         G      2      1      3       1      1      8      6      6      6
...     ...    ...    ...    ...     ...    ...    ...    ...    ...    ...
19995     D      2      2      3       3      2      7      7      7      6
19996     C      7     10      8       8      4      4      8      6      9
19997     T      6      9      6       7      5      6     11      3      7
19998     S      2      3      4       2      1      8      7      2      6
19999     A      4      9      6       6      2      9      5      3      1

       xybar  x2ybr  xy2br  x-ege  xegvy  y-ege  yegvx
0          6     10      8      0      8      0      8
1         13      3      9      2      8      4     10
2         10      3      7      3      7      3      9
3          4      4     10      6     10      2      8
4          6      5      9      1      7      5     10
...      ...    ...    ...    ...    ...    ...    ...
19995      6      6      4      2      8      3      7
19996     12      9     13      2      9      3      7
19997     11      9      5      2     12      2      4
19998     10      6      8      1      9      5      8
19999      8      1      8      2      7      2      8

[20000 rows x 17 columns]>
```

```
#Displaying particular row from df
df.loc[[4],:]
```

	lettr	x-box	y-box	width	height	onpix	x-bar	y-bar	x2bar	y2bar	xybar	x2ybr	xy2br	x-ege	xegvy	y-ege	yegvx
4	G	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10

```
x = df.drop("lettr", axis=1).values
y = df["lettr"].values
```

```
#Accessing single row of x using np.array
print(x[1])
x.shape
```

```
[ 5 12  3  7  2 10  5  5  4 13  3  9  2  8  4 10]
(20000, 16)
```

```
#Printing output(y) of corresponding input(x)
print(y[1])
y.shape
```

```
I
(20000,)
```

```
np.unique(y)
```

```
array(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
       'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'],
      dtype=object)
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

```
x_train[0]
```

```
array([ 6, 12,  7,  6,  3,  7,  7,  2,  3, 12,  5,  8,  5,  8,  0,  7])
```

```
def shape():
  print("Train Shape :",x_train.shape)
  print("Test Shape :",x_test.shape)
  print("y_train shape :",y_train.shape)
  print("y_test shape :",y_test.shape)
shape()
```

```
Train Shape : (16000, 16)
Test Shape : (4000, 16)
y_train shape : (16000,)
y_test shape : (4000,)
```

```
x_train[1]
```

```
array([ 5,  8,  6,  6,  4,  9,  7,  4,  6, 10,  3,  6,  2,  7,  5, 10])
```

```
y_test[1]
```

```
'Q'
```

```
class_names=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
```

Preprocessing

```
x_train = x_train/255
x_test = x_test/255
```

```
from sklearn.preprocessing import LabelEncoder
```

```
encoder = LabelEncoder()
y_train = encoder.fit_transform(y_train)
y_test = encoder.fit_transform(y_test)
a = encoder.fit_transform(y_train)
b = encoder.fit_transform(y_test)
```

```
np.unique(y_train)
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25])
```

```
np.unique(y_test)
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25])
```

Building our Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

```
model=Sequential()
```

```
model.add(Dense(512, activation='relu', input_shape=(16,)))
model.add(Dropout(0.2))
```

```
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
```

```
model.add(Dense(26, activation='softmax'))
```

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

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 512)               8704

 dropout_2 (Dropout)         (None, 512)               0

 dense_4 (Dense)             (None, 256)               131328

 dropout_3 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 26)                6682

=================================================================
Total params: 146714 (573.10 KB)
Trainable params: 146714 (573.10 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.fit(x_train, y_train, epochs=50, batch_size=128, verbose=1,validation_data=(x_test, y_test))
```

```
Epoch 1/50
125/125 [==============================] - 4s 5ms/step - loss: 3.1428 - accuracy: 0.1551 - val_loss: 2.8042 - val_accuracy: 0.2677
Epoch 2/50
125/125 [==============================] - 1s 6ms/step - loss: 2.3745 - accuracy: 0.3156 - val_loss: 2.0950 - val_accuracy: 0.4038
```

```
Epoch 3/50
125/125 [==============================] - 2s 12ms/step - loss: 1.9625 - accuracy: 0.4059 - val_loss: 1.8060 - val_accuracy: 0.4712
Epoch 4/50
125/125 [==============================] - 1s 10ms/step - loss: 1.7425 - accuracy: 0.4655 - val_loss: 1.6433 - val_accuracy: 0.5088
Epoch 5/50
125/125 [==============================] - 1s 4ms/step - loss: 1.6107 - accuracy: 0.5060 - val_loss: 1.5261 - val_accuracy: 0.5545
Epoch 6/50
125/125 [==============================] - 0s 3ms/step - loss: 1.5167 - accuracy: 0.5351 - val_loss: 1.4413 - val_accuracy: 0.5840
Epoch 7/50
125/125 [==============================] - 0s 3ms/step - loss: 1.4504 - accuracy: 0.5625 - val_loss: 1.3768 - val_accuracy: 0.6053
Epoch 8/50
125/125 [==============================] - 0s 3ms/step - loss: 1.3922 - accuracy: 0.5850 - val_loss: 1.3195 - val_accuracy: 0.6263
Epoch 9/50
125/125 [==============================] - 0s 3ms/step - loss: 1.3434 - accuracy: 0.5962 - val_loss: 1.2757 - val_accuracy: 0.6360
Epoch 10/50
125/125 [==============================] - 0s 4ms/step - loss: 1.2822 - accuracy: 0.6202 - val_loss: 1.2186 - val_accuracy: 0.6515
Epoch 11/50
125/125 [==============================] - 1s 6ms/step - loss: 1.2420 - accuracy: 0.6333 - val_loss: 1.1640 - val_accuracy: 0.6680
Epoch 12/50
125/125 [==============================] - 1s 6ms/step - loss: 1.1944 - accuracy: 0.6449 - val_loss: 1.1220 - val_accuracy: 0.6700
Epoch 13/50
125/125 [==============================] - 0s 4ms/step - loss: 1.1566 - accuracy: 0.6555 - val_loss: 1.0959 - val_accuracy: 0.6783
Epoch 14/50
125/125 [==============================] - 0s 3ms/step - loss: 1.1249 - accuracy: 0.6662 - val_loss: 1.0504 - val_accuracy: 0.6915
Epoch 15/50
125/125 [==============================] - 0s 3ms/step - loss: 1.0847 - accuracy: 0.6791 - val_loss: 1.0200 - val_accuracy: 0.6998
Epoch 16/50
125/125 [==============================] - 0s 3ms/step - loss: 1.0621 - accuracy: 0.6846 - val_loss: 0.9763 - val_accuracy: 0.7175
Epoch 17/50
125/125 [==============================] - 0s 3ms/step - loss: 1.0262 - accuracy: 0.6932 - val_loss: 0.9498 - val_accuracy: 0.7215
Epoch 18/50
125/125 [==============================] - 0s 3ms/step - loss: 0.9990 - accuracy: 0.7028 - val_loss: 0.9331 - val_accuracy: 0.7310
Epoch 19/50
125/125 [==============================] - 0s 3ms/step - loss: 0.9729 - accuracy: 0.7066 - val_loss: 0.9037 - val_accuracy: 0.7372
Epoch 20/50
125/125 [==============================] - 0s 4ms/step - loss: 0.9450 - accuracy: 0.7182 - val_loss: 0.8833 - val_accuracy: 0.7437
Epoch 21/50
125/125 [==============================] - 1s 4ms/step - loss: 0.9277 - accuracy: 0.7230 - val_loss: 0.8482 - val_accuracy: 0.7525
Epoch 22/50
125/125 [==============================] - 0s 3ms/step - loss: 0.9066 - accuracy: 0.7258 - val_loss: 0.8397 - val_accuracy: 0.7527
Epoch 23/50
125/125 [==============================] - 0s 3ms/step - loss: 0.8859 - accuracy: 0.7346 - val_loss: 0.8070 - val_accuracy: 0.7722
Epoch 24/50
125/125 [==============================] - 0s 3ms/step - loss: 0.8651 - accuracy: 0.7398 - val_loss: 0.8007 - val_accuracy: 0.7673
Epoch 25/50
125/125 [==============================] - 0s 3ms/step - loss: 0.8439 - accuracy: 0.7450 - val_loss: 0.7901 - val_accuracy: 0.7590
Epoch 26/50
125/125 [==============================] - 0s 3ms/step - loss: 0.8327 - accuracy: 0.7496 - val_loss: 0.7512 - val_accuracy: 0.7785
Epoch 27/50
125/125 [==============================] - 0s 3ms/step - loss: 0.8138 - accuracy: 0.7545 - val_loss: 0.7337 - val_accuracy: 0.7822
Epoch 28/50
125/125 [==============================] - 0s 3ms/step - loss: 0.7916 - accuracy: 0.7623 - val_loss: 0.7165 - val_accuracy: 0.7835
Epoch 29/50
125/125 [==============================] - 0s 3ms/step - loss: 0.7735 - accuracy: 0.7651 - val_loss: 0.7013 - val_accuracy: 0.7838
```

Testing our Model

```
predictions = model.predict(x_test)
```

```
125/125 [==============================] - 0s 2ms/step
```

```
index=9
print(predictions[index])
final_value=np.argmax(predictions[index])
print("Actual label :",y_test[index])
print("Predicted label :",final_value)
print("Class (A-Z) :",class_names[final_value])
```

```
[1.1053467e-03 7.3635431e-09 2.0758957e-07 9.5543033e-04 2.1689908e-10
 5.1755618e-05 7.6691400e-08 3.1211769e-02 1.6342166e-07 1.8307472e-05
 8.3997725e-07 2.1503110e-06 7.0475362e-04 8.6267394e-01 3.8787076e-04
 3.4749035e-05 1.6144359e-06 3.3106980e-07 3.1333761e-07 1.8954182e-03
 8.9017212e-02 6.4201388e-03 5.2498756e-03 1.7106619e-04 9.6595177e-05
 1.2424348e-12]
Actual label : 13
Predicted label : 13
Class (A-Z) : N
```