Ex. No: 3 Y.V.Ojus
12-2-24 3122 21 5001 125

Ex3 - Handwritten character recognition using neural networks

Colab Link:

Colab Link

Aim:

To understand neural networks and perform the task of handwritten character recognition

Code & Output:

Import Dependencies

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPool2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import cv2
import time
import urllib.request
from PIL import Image
```

Reading And Sampling Data

```
df = pd.read_csv('english.csv')
df.head()
```

	image	label
0	Img/img001-001.png	0
1	Img/img001-002.png	0
2	Img/img001-003.png	0
3	Img/img001-004.png	0
4	Img/img001-005.png	0

df.columns

```
Index(['image', 'label'], dtype='object')
```

df.index

```
RangeIndex(start=0, stop=3410, step=1)
```

```
im = Image.open(df.iloc[np.random.randint(0,3410),0])
im.show()
```

Pre Processing

Handling NULL Values

df['image'].isnull().sum()

```
0
```

df['label'].isnull().sum()

```
0
```

Ex. No: 3 Y.V.Ojus
12-2-24 3122 21 5001 125

URL Image Read Function with OpenCV

```
def read_url_pic(x):
    image_url = x

    with urllib.request.urlopen(image_url) as url:
        s = url.read()

    arr = np.asarray(bytearray(s), dtype=np.uint8)
    image = cv2.imdecode(arr, -1)
    return image
```

Plot Image from URL with Matplotlib

```
def plot_url_pic(x):
    image_url = x

    with urllib.request.urlopen(image_url) as url:
        s = url.read()

    arr = np.asarray(bytearray(s), dtype=np.uint8)
    image = cv2.imdecode(arr, -1)

    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.show()
```

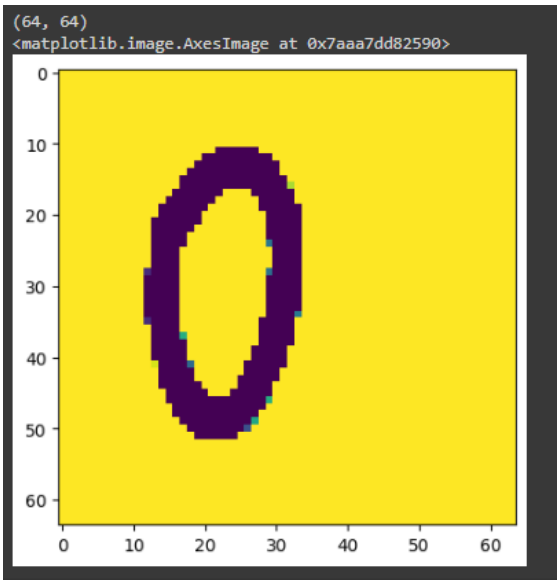
Convert Image to Array with OpenCV

```
def img_to_arr(x):
    img = cv2.imread(x)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img,(64,64))
    return img
```

Ex. No: 3 Y.V.Ojus
12-2-24 3122 21 5001 125

Displaying Image Array Shape and Visualization

```
print(img_to_arr(df['image'][0]).shape)
plt.imshow(img_to_arr(df['image'][0]))
```



Parallel Image Processing with Multiprocessing

```
start_time = time.time()
import multiprocessing

with multiprocessing.Pool(4) as p:
    images = p.map(img_to_arr, df['image'])
end_time = time.time()
print("with multiprocessing:", end_time-start_time)
```

```
with multiprocessing: 229.7383852005005
```

Reshape Image Array for Deep Learning Model Input

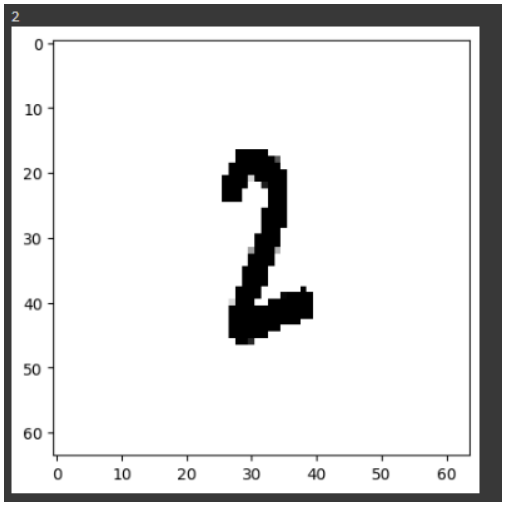
```
x = np.array(images).reshape(3410, 64, 64,1)
x.shape
```

```
(3410, 64, 64, 1)
```

Ex. No: 3 Y.V.Ojus
12-2-24 3122 21 5001 125

Visualizing Image with Corresponding Label

```
img = x[119].reshape((64,64))
plt.imshow(img,cmap='gray')
print(df['label'].iloc[119])
```



Label Encoding for Classification

```
y = df['label']
le = LabelEncoder()
y_label = le.fit_transform(y)
```

Splitting Data into Training and Testing Sets

```
train_images,test_images,train_labels,test_labels =
train_test_split(x,y_label,test_size=0.2,random_state=42)
train_images = train_images/255.0
test_images = test_images/255.0
```

Ex. No: 3 Y.V.Ojus
12-2-24 3122 21 5001 125

Convolutional Neural Network Model Training

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
model = Sequential()
model.add(Conv2D(512, (5, 5), activation='relu', input_shape=(64, 64, 1)))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(62, activation='softmax'))
```

```
Epoch 1/20
86/86 [==============================] - 6s 41ms/step - loss: 4.1317 - accuracy: 0.0172 - val_loss: 4.1273 - val_accuracy: 0.0191
Epoch 2/20
86/86 [==============================] - 3s 37ms/step - loss: 4.0511 - accuracy: 0.0330 - val_loss: 3.8516 - val_accuracy: 0.0777
Epoch 3/20
86/86 [==============================] - 3s 40ms/step - loss: 3.7260 - accuracy: 0.0806 - val_loss: 3.2224 - val_accuracy: 0.1921
Epoch 4/20
86/86 [==============================] - 4s 41ms/step - loss: 3.2124 - accuracy: 0.1675 - val_loss: 2.5603 - val_accuracy: 0.3666
Epoch 5/20
86/86 [==============================] - 3s 39ms/step - loss: 2.5948 - accuracy: 0.3050 - val_loss: 1.8749 - val_accuracy: 0.4897
Epoch 6/20
86/86 [==============================] - 3s 38ms/step - loss: 2.1333 - accuracy: 0.4054 - val_loss: 1.4942 - val_accuracy: 0.5909
Epoch 7/20
86/86 [==============================] - 3s 40ms/step - loss: 1.7806 - accuracy: 0.4703 - val_loss: 1.1092 - val_accuracy: 0.6525
Epoch 8/20
86/86 [==============================] - 3s 37ms/step - loss: 1.5071 - accuracy: 0.5403 - val_loss: 0.9838 - val_accuracy: 0.7199
Epoch 9/20
86/86 [==============================] - 3s 37ms/step - loss: 1.2936 - accuracy: 0.6008 - val_loss: 0.9504 - val_accuracy: 0.7111
Epoch 10/20
86/86 [==============================] - 3s 39ms/step - loss: 1.1632 - accuracy: 0.6499 - val_loss: 0.8682 - val_accuracy: 0.7419
Epoch 11/20
86/86 [==============================] - 3s 39ms/step - loss: 1.0106 - accuracy: 0.6859 - val_loss: 0.7958 - val_accuracy: 0.7639
Epoch 12/20
86/86 [==============================] - 3s 38ms/step - loss: 0.8861 - accuracy: 0.7137 - val_loss: 0.7585 - val_accuracy: 0.7713
Epoch 13/20
86/86 [==============================] - 3s 38ms/step - loss: 0.8085 - accuracy: 0.7302 - val_loss: 0.7352 - val_accuracy: 0.7727
Epoch 14/20
86/86 [==============================] - 3s 38ms/step - loss: 0.7586 - accuracy: 0.7588 - val_loss: 0.7822 - val_accuracy: 0.7625
Epoch 15/20
86/86 [==============================] - 3s 39ms/step - loss: 0.7127 - accuracy: 0.7625 - val_loss: 0.7015 - val_accuracy: 0.7845
Epoch 16/20
86/86 [==============================] - 3s 38ms/step - loss: 0.5929 - accuracy: 0.8002 - val_loss: 0.6656 - val_accuracy: 0.7977
Epoch 17/20
86/86 [==============================] - 3s 37ms/step - loss: 0.5764 - accuracy: 0.8109 - val_loss: 0.6602 - val_accuracy: 0.7962
Epoch 18/20
86/86 [==============================] - 3s 37ms/step - loss: 0.5377 - accuracy: 0.8196 - val_loss: 0.6942 - val_accuracy: 0.7918
Epoch 19/20
86/86 [==============================] - 3s 39ms/step - loss: 0.5230 - accuracy: 0.8284 - val_loss: 0.6490 - val_accuracy: 0.8006
Epoch 20/20
86/86 [==============================] - 3s 38ms/step - loss: 0.4320 - accuracy: 0.8614 - val_loss: 0.6829 - val_accuracy: 0.7991
<keras.src.callbacks.History at 0x7bc9a04ad030>
```

Ex. No: 3 Y.V.Ojus
12-2-24 3122 21 5001 125

Display Model Summary

model.summary()

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 60, 60, 512)       13312

 max_pooling2d_6 (MaxPoolin  (None, 30, 30, 512)       0
 g2D)

 conv2d_8 (Conv2D)           (None, 28, 28, 256)       1179904

 max_pooling2d_7 (MaxPoolin  (None, 14, 14, 256)       0
 g2D)

 conv2d_9 (Conv2D)           (None, 12, 12, 256)       590080

 max_pooling2d_8 (MaxPoolin  (None, 6, 6, 256)         0
 g2D)

 flatten_2 (Flatten)         (None, 9216)              0

 dense_6 (Dense)             (None, 256)               2359552

 dropout (Dropout)           (None, 256)               0

 dense_7 (Dense)             (None, 128)               32896

 dropout_1 (Dropout)         (None, 128)               0

 dense_8 (Dense)             (None, 62)                7998

=================================================================
Total params: 4183742 (15.96 MB)
Trainable params: 4183742 (15.96 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Inverse Transform Predicted and Actual Labels

predicted_labels = le.inverse_transform(model.predict(test_images).argmax(axis=1))
actual_labels = le.inverse_transform(test_labels)

```
22/22 [==============================] - 1s 13ms/step
```

Ex. No: 3 Y.V.Ojus
12-2-24 3122 21 5001 125

Display Predicted and Actual Labels for a Specific Index

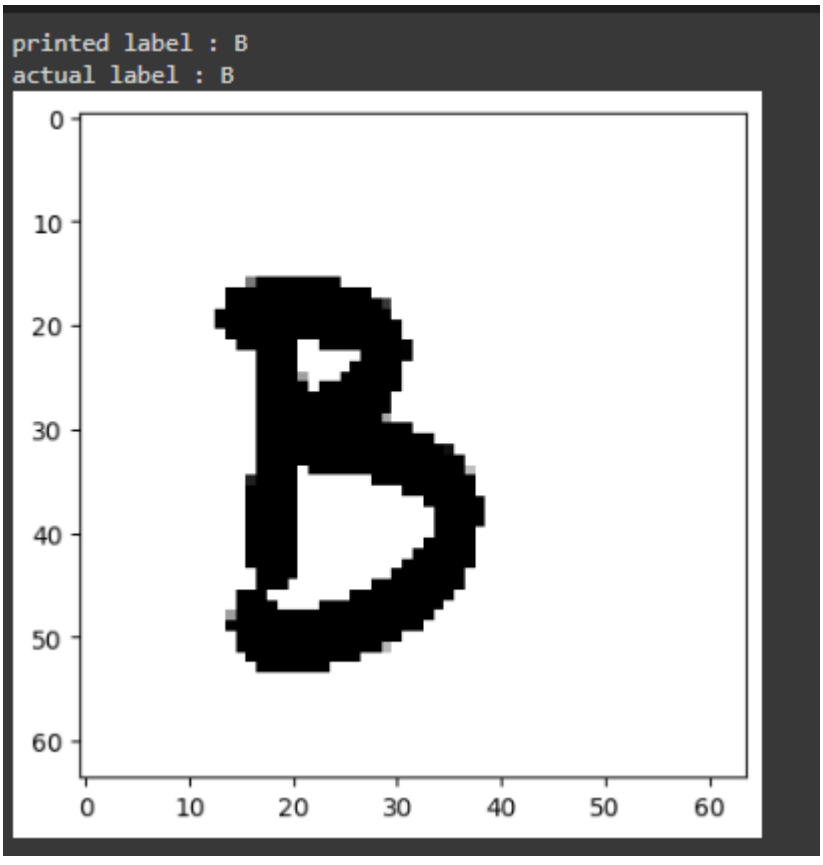
```
print('predicted label is ;',predicted_labels[67])
print('Actual Label is :',actual_labels[67])
```

```
predicted label is ; j
Actual Label is : j
```

Displaying Predicted and Actual Labels for a Specific Image

```
# number of image to be predicted
# change the value of I
i = 450

plt.imshow(test_images[i],cmap='gray')
print('printed label :',predicted_labels[i])
print('actual label :', actual_labels[i])
```



Ex. No: 3 Y.V.Ojus
12-2-24 3122 21 5001 125

Evaluate Model Performance with Various Metrics

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix


y_pred = model.predict(test_images)
y_pred_labels = np.argmax(y_pred, axis=1)

# Calculate evaluation metrics
accuracy = accuracy_score(test_labels, y_pred_labels)
precision = precision_score(test_labels, y_pred_labels, average='weighted')
recall = recall_score(test_labels, y_pred_labels, average='weighted')
f1 = f1_score(test_labels, y_pred_labels, average='weighted')
confusion_mat = confusion_matrix(test_labels, y_pred_labels)

# Print or use the evaluation metrics as needed
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(confusion_mat)
```

```
22/22 [==============================] - 0s 12ms/step
Accuracy: 0.7991202346041055
Precision: 0.8160896386154118
Recall: 0.7991202346041055
F1 Score: 0.7966589813430557
Confusion Matrix:
[[ 8  0  0 ...  0  0  0]
 [ 0  5  0 ...  0  0  0]
 [ 0  0  8 ...  0  0  0]
 ...
 [ 0  0  0 ... 12  0  2]
 [ 0  0  0 ...  0  8  0]
 [ 0  0  0 ...  0  0  9]]
```