```
from google.colab import drive
drive.mount('/gdrive')
%cd /gdrive
```

```
Drive already mounted at /gdrive; to attempt to forcibly remount, call drive.mount("/gdr
/gdrive
```

```
ls
```

```
MyDrive/  Shareddrives/
```

```
cd /gdrive/My Drive/Breed Classification
```

```
/gdrive/My Drive/Breed Classification
```

```
ls
```

```
SheepFaceImages/
```

```
cd /gdrive/My Drive/Breed Classification/SheepFaceImages
```

```
/gdrive/My Drive/Breed Classification/SheepFaceImages
```

```
ls
```

```
 Marino/  'Poll Dorset'/   Suffolk/  'White Suffolk'/
```

```
import os

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


from skimage import io
from skimage.color import rgb2gray
from skimage.transform import rescale, resize, downscale_local_mean
from skimage import morphology


from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


from keras.models import Sequential
```

```
from keras.layers import Dense, Conv2D, Flatten, Dropout
from keras.utils import to_categorical


Marino = "/gdrive/My Drive/Breed Classification/SheepFaceImages/Marino/"
WhiteSuffolk = "/gdrive/My Drive/Breed Classification/SheepFaceImages/White Suffolk"
PollDorset = "/gdrive/My Drive/Breed Classification/SheepFaceImages/Poll Dorset"
Suffolk = "/gdrive/My Drive/Breed Classification/SheepFaceImages/Suffolk"


ls
```

```
 Marino/  'Poll Dorset'/   Suffolk/  'White Suffolk'/
```

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))


os.listdir("/gdrive/My Drive/Breed Classification/SheepFaceImages")
```

```
['White Suffolk', 'Marino', 'Suffolk', 'Poll Dorset']
```

```
from tqdm import tqdm
x = []
y = []

def create_dataset(dirname,breedname):
    for i in tqdm(os.listdir(dirname)):
        path = os.path.join(dirname,i)
        try:
            img = cv2.imread(path)
            img = cv2.resize(img,(150,150))
        except:
            continue

        x.append(img)
        y.append(breedname)
    return x,y


x,y = create_dataset(Marino,"Marino")
x,y = create_dataset(WhiteSuffolk,"White Suffolk")
x,y = create_dataset(PollDorset,"Poll Dorset")
x,y = create_dataset(Suffolk,"Suffolk")
```

```
100%|██████████| 420/420 [00:00<00:00, 202856.71it/s]
100%|██████████| 420/420 [00:00<00:00, 208055.71it/s]
100%|██████████| 420/420 [00:00<00:00, 138241.21it/s]
100%|██████████| 420/420 [00:00<00:00, 92521.41it/s]
```

```
BASE_PATH = '/gdrive/My Drive/Breed Classification/SheepFaceImages/'

sheep_data = {}

image_filename_index = 0

for sheep_bread in os.listdir(BASE_PATH):
    print(sheep_bread)
    BREAD_PATH = BASE_PATH + '/' + sheep_bread
    for image_filename in os.listdir(BREAD_PATH):
        image_path = BREAD_PATH + '/' + image_filename
        #read the image
        sheep = io.imread(image_path)
        # convert to grayscale
        sheep = rgb2gray(sheep)
        # resize
        sheep = resize(sheep, (90,78), anti_aliasing=False)
        # reshape
        sheep = sheep.reshape(sheep.shape[0] * sheep.shape[1])
        sheep = np.append(sheep, str(sheep_bread))
        sheep_data[image_filename_index] = sheep
        image_filename_index = image_filename_index + 1

column_names = ['pixel_' + str(i) for i in range(0,7020)]
column_names = np.append(column_names, 'bread')
df = pd.DataFrame.from_dict(sheep_data, orient='index', columns=column_names)
df.to_csv('Sheep_bread_dataset.csv', index=None)
```

```
 White Suffolk
 Marino
 Suffolk
 Poll Dorset
```

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

	pixel_0	pixel_1	pixel_2	pixel_3	pixel_4	pixel_5	pixel_6	pixel_7	pixel_8
0	0.535896	0.735112	0.596092	0.715351	0.715874	0.807682	0.770199	0.787922	0.663553

```
X = df.drop(['bread'], axis=1).values

encoder = LabelEncoder()
df['bread'] = encoder.fit_transform(df['bread'])
Y = to_categorical(df['bread'])


X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.3, shuffle=True, stratif


X_train = X_train.reshape(len(X_train),90,78,1)
X_test = X_test.reshape(len(X_test),90,78,1)


#create model
model = Sequential()
#add model layers
model.add(Conv2D(filters=64, kernel_size=3,strides=(2,1), padding='same', activation='relu',
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(4, activation='softmax'))

# compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 45, 78, 64)        640
_________________________________________________________________
flatten_4 (Flatten)          (None, 224640)            0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                14377024
_________________________________________________________________
dense_11 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_12 (Dense)             (None, 4)                 132
=================================================================
Total params: 14,379,876
Trainable params: 14,379,876
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(X_train, y_train, batch_size=128, epochs=20)
```

```
Epoch 1/20
10/10 [==============================] - 7s 559ms/step - loss: 1.6770 - accuracy: 0.305
```

```
Epoch 2/20
10/10 [==============================] - 6s 556ms/step - loss: 1.1311 - accuracy: 0.5118
Epoch 3/20
10/10 [==============================] - 6s 554ms/step - loss: 0.9243 - accuracy: 0.5866
Epoch 4/20
10/10 [==============================] - 5s 534ms/step - loss: 0.7427 - accuracy: 0.7021
Epoch 5/20
10/10 [==============================] - 6s 546ms/step - loss: 0.6493 - accuracy: 0.7059
Epoch 6/20
10/10 [==============================] - 5s 545ms/step - loss: 0.4813 - accuracy: 0.8434
Epoch 7/20
10/10 [==============================] - 5s 534ms/step - loss: 0.4222 - accuracy: 0.8434
Epoch 8/20
10/10 [==============================] - 5s 532ms/step - loss: 0.4109 - accuracy: 0.8689
Epoch 9/20
10/10 [==============================] - 5s 541ms/step - loss: 0.2756 - accuracy: 0.9403
Epoch 10/20
10/10 [==============================] - 5s 544ms/step - loss: 0.2302 - accuracy: 0.9467
Epoch 11/20
10/10 [==============================] - 5s 539ms/step - loss: 0.1722 - accuracy: 0.9628
Epoch 12/20
10/10 [==============================] - 5s 529ms/step - loss: 0.1409 - accuracy: 0.9726
Epoch 13/20
10/10 [==============================] - 5s 541ms/step - loss: 0.1226 - accuracy: 0.9826
Epoch 14/20
10/10 [==============================] - 5s 539ms/step - loss: 0.1095 - accuracy: 0.9835
Epoch 15/20
10/10 [==============================] - 5s 534ms/step - loss: 0.1111 - accuracy: 0.9778
Epoch 16/20
10/10 [==============================] - 5s 524ms/step - loss: 0.0791 - accuracy: 0.9915
Epoch 17/20
10/10 [==============================] - 5s 527ms/step - loss: 0.0524 - accuracy: 0.9973
Epoch 18/20
10/10 [==============================] - 5s 531ms/step - loss: 0.0484 - accuracy: 0.9973
Epoch 19/20
10/10 [==============================] - 6s 547ms/step - loss: 0.0279 - accuracy: 0.9989
Epoch 20/20
10/10 [==============================] - 6s 553ms/step - loss: 0.0246 - accuracy: 0.9997
<tensorflow.python.keras.callbacks.History at 0x7fdac1c3d748>
```

```
y_train_pred = model.predict_classes(X_train)
y_test_pred = model.predict_classes(X_test)
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/sequential.py:450
  warnings.warn('`model.predict_classes()` is deprecated and '
```

```
def normalize_value(predictions):
    normalized = []
    for prediction in predictions:
        result = 0
        for i in range(0,len(prediction)):
            result += i * prediction[i]
```

```
            normalized.append(int(result))
        return normalized

    normalized_y_train = normalize_value(y_train)
    normalized_y_test = normalize_value(y_test)


    train_conf_matrix = np.zeros(16).reshape(4,4)

    for index in range(0,len(normalized_y_train)):
        train_value = normalized_y_train[index]
        predicted_train_value = y_train_pred[index]
        train_conf_matrix[train_value][predicted_train_value] += 1

    test_conf_matrix = np.zeros(16).reshape(4,4)

    for index in range(0,len(normalized_y_test)):
        test_value = normalized_y_test[index]
        predicted_test_value = y_test_pred[index]
        test_conf_matrix[test_value][predicted_test_value] += 1


    train_conf_matrix_df = pd.DataFrame(train_conf_matrix)
    train_conf_matrix_df.columns = ['Marino', 'Poll Dorset', 'Suffolk' , 'White Suffolk']
    train_conf_matrix_df.index = ['Marino', 'Poll Dorset', 'Suffolk' , 'White Suffolk']
    train_conf_matrix_df
```

	Marino	Poll Dorset	Suffolk	White Suffolk
Marino	294.0	0.0	0.0	0.0
Poll Dorset	0.0	294.0	0.0	0.0
Suffolk	0.0	0.0	294.0	0.0
White Suffolk	0.0	0.0	0.0	294.0

```
    test_conf_matrix_df = pd.DataFrame(test_conf_matrix)
    test_conf_matrix_df.columns = ['Marino', 'Poll Dorset', 'Suffolk' , 'White Suffolk']
    test_conf_matrix_df.index = ['Marino', 'Poll Dorset', 'Suffolk' , 'White Suffolk']
    test_conf_matrix_df
```

	Marino	Poll Dorset	Suffolk	White Suffolk
Marino	97.0	18.0	0.0	11.0
Poll Dorset	7.0	116.0	2.0	1.0
Suffolk	3.0	0.0	123.0	0.0
White Suffolk	12.0	11.0	0.0	103.0

```
    print("Accuracy Train : 99.97")
    print('Accuracy test : %.3f' %(accuracy_score(normalized_y_test,y_test_pred)) )
```

```
print('Accuracy test : %.3f' %(accuracy_score(normalized_y_test,y_test_pred)) )
```

```
Accuracy Train : 99.97
Accuracy test : 0.871
```