Pet Classification using CNN model :

In [2]:
```python
### Import required libraries :
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import random
from sklearn.metrics import classification_report, confusion_matrix

import tensorflow as tf
import keras
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout, BatchNormalization
from keras.models import Sequential

import os
import cv2

from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
```

Data Augmentation :

In [ ]:
```python
### Instantiating ImageDataGenerator :
train_datagen = ImageDataGenerator(
    rotation_range = 40,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True,
    fill_mode = 'nearest'
)

### Defining imageGenerator function to create new images :
def imageGenerator(pc_directory, imagesize, save_directory):

    train_generator = train_datagen.flow_from_directory(

    pc_directory,
    target_size = (imagesize, imagesize),
    batch_size = 20,
    #class_mode = 'binary',
    save_to_dir = save_directory,
    save_prefix = 'cat',
    save_format = 'jpg'
)

    print(train_generator.class_indices)

    ## Iterarter to produce new images :
    for i in range(200):
        images, labels = next(train_generator)
```

In [ ]:
```python
pc_directory = r'C:\Users\hp\OneDrive\Desktop\AI\Deep Learning\Datasets\Simplilearn project datasets\1577957291_deeplearningwithkerasandtensorflow project3\da
imagesize = 128
save_directory = r'C:\Users\hp\OneDrive\Desktop\AI\Deep Learning\Datasets\Simplilearn project datasets\1577957291_deeplearningwithkerasandtensorflow project3\
imageGenerator(pc_directory, imagesize, save_directory)
```

In [ ]:
```python
pc_directory = r'C:\Users\hp\OneDrive\Desktop\AI\Deep Learning\Datasets\Simplilearn project datasets\1577957291_deeplearningwithkerasandtensorflow project3\da
imagesize = 128
save_directory = r'C:\Users\hp\OneDrive\Desktop\AI\Deep Learning\Datasets\Simplilearn project datasets\1577957291_deeplearningwithkerasandtensorflow project3\
imageGenerator(pc_directory, imagesize, save_directory)
```

In [19]:
```python
### Function to load data from the pc :
def load_data(DATA_DIR, CATEGORIES, directory):
    data = []

    for category in CATEGORIES:
        path = os.path.join(DATA_DIR, directory)
        path = os.path.join(path, category)
        class_names = CATEGORIES.index(category)

        for image in os.listdir(path):
            img_array = cv2.imread(os.path.join(path, image))
            data.append([img_array, class_names])

    return data
```

In [20]:
```python
### loading training data to the list :
DATA_DIR = r'C:\Users\hp\OneDrive\Desktop\AI\Deep Learning\Datasets\Simplilearn project datasets\1577957291_deeplearningwithkerasandtensorflow project3\data'
CATEGORIES_2 = ['cats', 'dogs']
directory = 'train'
training_data = load_data(DATA_DIR, CATEGORIES_2, directory)
type(training_data)
```

Out[20]:
```
list
```

In [21]:
```python
### loading testing data to the list :
directory = 'test'
CATEGORIES = ['cats', 'dogs']
testing_data = load_data(DATA_DIR, CATEGORIES, directory)
```

In [22]:
```python
### Shuffle the training and testing data :
def shuffle_data(data):
    random.shuffle(data)

shuffle_data(training_data)
shuffle_data(testing_data)

## Checking whether the data is shuffled or not in train_images :
print('Train_labels :')
for img in training_data[:10]:
    print(img[1])

## Checking whether the data is shuffled or not in test_images:
print('Test_labels :')
for img in testing_data[:10]:
    print(img[1])
len(training_data)
```

```
Train_labels :
0
0
1
1
1
1
1
0
0
1
Test_labels :
1
0
0
0
0
1
0
1
1
0
```

Out[22]:
```
8040
```

In [23]:
```python
### No.of images in train and test datasets :
print('training data length :', len(training_data))
print('testing data length :', len(testing_data))
```

```
training data length : 8040
testing data length : 20
```

In [24]:
```python
### Splitting the data into train and test splits :
X_train = []
y_train = []
X_test = []
y_test = []

for feature, label in training_data:
    X_train.append(feature)
    y_train.append(label)

for feature, label in testing_data:
    X_test.append(feature)
    y_test.append(label)

print('No.of images in X_train :', len(X_train))
print('No.of images in y_train :', len(y_train))
print('=' * 40)
print('No.of labels in X_test :', len(X_test))
print('No.of labels in y_test :', len(y_test))
```

```
No.of images in X_train : 8040
No.of images in y_train : 8040
========================================
No.of labels in X_test : 20
No.of labels in y_test : 20
```

In [25]:
```python
### Resize all images to the same size :
def resize_image(data):
    resized_data = []
    image_size = 128
    for image in data:
        temp_img = cv2.resize(image, (image_size, image_size), interpolation = cv2.INTER_NEAREST)
        resized_data.append(temp_img)

    return resized_data

X_train_resized = resize_image(X_train)
X_test_resized = resize_image(X_test)
```

In [26]:
```python
### list to array conversion :
def array_conv(data):
    data = np.asarray(data)
    return data

X_train_arr = array_conv(X_train_resized)
X_test_arr = array_conv(X_test_resized)
y_train_arr = array_conv(y_train)
y_test_arr = array_conv(y_test)
print(type(X_train_resized))
print(type(X_test_resized))
print('=' * 40)
print(type(y_train))
print(type(y_test))
```

```
<class 'list'>
<class 'list'>
========================================
<class 'list'>
<class 'list'>
```

In [27]:
```python
X_train_arr.shape
```

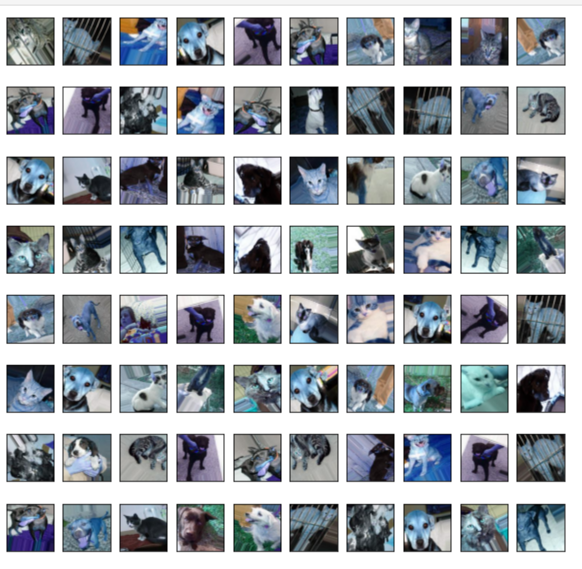
Out[27]:
```
(8040, 128, 128, 3)
```

In [28]:
```python
### Scaling data :
def scaling_data(data):
    data = data / 255.0
    return data

X_train_scaled = scaling_data(X_train_arr)
X_test_scaled = scaling_data(X_test_arr)
```

In [29]:
```python
X_train_scaled.shape
```

Out[29]:
```
(8040, 128, 128, 3)
```

In [30]:
```python
### Plotting some of the images :
plt.figure(figsize = (10, 10))
for i in range(80):
    plt.subplot(8, 10, i + 1)
    plt.imshow(X_train[i])
    plt.xticks([])
    plt.yticks([])
    plt.xlabel(CATEGORIES[y_train[i]], color = 'white')
```



In [31]:
```python
### Checking shapes of the train and test sets :
print('X_train scaled shape :', X_train_scaled.shape)
print('X_test scaled shape :', X_test_scaled.shape)
print('=' * 40)
print('y_train shape :', y_train_arr.shape)
print('y_test shape :', y_test_arr.shape)
```

```
X_train scaled shape : (8040, 128, 128, 3)
X_test scaled shape : (20, 128, 128, 3)
========================================
y_train shape : (8040,)
y_test shape : (20,)
```

In [ ]:

CNN :

In [32]:
```python
### CNN Model :
model = Sequential()

model.add(Conv2D(32, (5, 5), input_shape = X_train_scaled.shape[1 :], activation = 'relu'))
model.add(MaxPooling2D((2, 2), (2, 2)))
model.add(Conv2D(64, (5, 5), activation = 'relu'))
model.add(MaxPooling2D((2, 2), (2, 2)))

model.add(Flatten())
model.add(Dense(32, activation = 'relu'))
model.add(Dropout(0.4))
model.add(Dense(2, activation = 'softmax'))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 124, 124, 32)      2432
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 62, 62, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 58, 58, 64)        51264
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 29, 29, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 53824)             0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1722400
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 66
=================================================================
Total params: 1,776,162
Trainable params: 1,776,162
Non-trainable params: 0
_________________________________________________________________
```

In [33]:
```python
### Compiling the model :
model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy'])
```

In [34]:
```python
### Fitting the model :
model.fit(X_train_scaled, y_train_arr, epochs = 50, validation_split = 0.1)
```

```
Epoch 1/50
227/227 [==============================] - 238s 987ms/step - loss: 0.7204 - accuracy: 0.5701 - val_loss: 0.4789 - val_accuracy: 0.7239
Epoch 2/50
227/227 [==============================] - 215s 946ms/step - loss: 0.4541 - accuracy: 0.7836 - val_loss: 0.2546 - val_accuracy: 0.8806
Epoch 3/50
227/227 [==============================] - 219s 963ms/step - loss: 0.2191 - accuracy: 0.9089 - val_loss: 0.1640 - val_accuracy: 0.9465
Epoch 4/50
227/227 [==============================] - 215s 949ms/step - loss: 0.1044 - accuracy: 0.9588 - val_loss: 0.0569 - val_accuracy: 0.9776
Epoch 5/50
227/227 [==============================] - 213s 938ms/step - loss: 0.1276 - accuracy: 0.9619 - val_loss: 0.0633 - val_accuracy: 0.9764
Epoch 6/50
227/227 [==============================] - 216s 952ms/step - loss: 0.0390 - accuracy: 0.9863 - val_loss: 0.2272 - val_accuracy: 0.9279
Epoch 7/50
227/227 [==============================] - 219s 967ms/step - loss: 0.0875 - accuracy: 0.9679 - val_loss: 0.0341 - val_accuracy: 0.9888
Epoch 8/50
227/227 [==============================] - 213s 939ms/step - loss: 0.0216 - accuracy: 0.9906 - val_loss: 0.0632 - val_accuracy: 0.9764
Epoch 9/50
227/227 [==============================] - 214s 942ms/step - loss: 0.0302 - accuracy: 0.9886 - val_loss: 0.0411 - val_accuracy: 0.9838
Epoch 10/50
227/227 [==============================] - 212s 933ms/step - loss: 0.0360 - accuracy: 0.9870 - val_loss: 0.0704 - val_accuracy: 0.9776
Epoch 11/50
227/227 [==============================] - 207s 913ms/step - loss: 0.0218 - accuracy: 0.9921 - val_loss: 0.1261 - val_accuracy: 0.9764
Epoch 12/50
227/227 [==============================] - 207s 913ms/step - loss: 0.0847 - accuracy: 0.9767 - val_loss: 0.0474 - val_accuracy: 0.9863
Epoch 13/50
227/227 [==============================] - 207s 912ms/step - loss: 0.0134 - accuracy: 0.9957 - val_loss: 0.0247 - val_accuracy: 0.9913
Epoch 14/50
227/227 [==============================] - 209s 919ms/step - loss: 0.0180 - accuracy: 0.9940 - val_loss: 0.0529 - val_accuracy: 0.9813
Epoch 15/50
227/227 [==============================] - 208s 916ms/step - loss: 0.0448 - accuracy: 0.9876 - val_loss: 0.0390 - val_accuracy: 0.9838
Epoch 16/50
227/227 [==============================] - 26174s 116s/step - loss: 0.0250 - accuracy: 0.9926 - val_loss: 0.0185 - val_accuracy: 0.9913
Epoch 17/50
227/227 [==============================] - 223s 983ms/step - loss: 0.0154 - accuracy: 0.9957 - val_loss: 0.0523 - val_accuracy: 0.9838
Epoch 18/50
227/227 [==============================] - 214s 944ms/step - loss: 0.0095 - accuracy: 0.9961 - val_loss: 0.0413 - val_accuracy: 0.9900
Epoch 19/50
227/227 [==============================] - 218s 962ms/step - loss: 0.0148 - accuracy: 0.9956 - val_loss: 0.0253 - val_accuracy: 0.9900
Epoch 20/50
227/227 [==============================] - 303s 1s/step - loss: 0.0174 - accuracy: 0.9955 - val_loss: 0.0291 - val_accuracy: 0.9925
Epoch 21/50
227/227 [==============================] - 311s 1s/step - loss: 0.0060 - accuracy: 0.9987 - val_loss: 0.0453 - val_accuracy: 0.9913
Epoch 22/50
227/227 [==============================] - 302s 1s/step - loss: 0.0057 - accuracy: 0.9978 - val_loss: 0.0947 - val_accuracy: 0.9689
Epoch 23/50
227/227 [==============================] - 265s 1s/step - loss: 0.0297 - accuracy: 0.9904 - val_loss: 0.0468 - val_accuracy: 0.9888
Epoch 24/50
227/227 [==============================] - 281s 1s/step - loss: 0.0114 - accuracy: 0.9963 - val_loss: 0.0234 - val_accuracy: 0.9888
Epoch 25/50
227/227 [==============================] - 302s 1s/step - loss: 0.0096 - accuracy: 0.9973 - val_loss: 0.0439 - val_accuracy: 0.9863
Epoch 26/50
227/227 [==============================] - 317s 1s/step - loss: 0.0220 - accuracy: 0.9928 - val_loss: 0.0282 - val_accuracy: 0.9925
Epoch 27/50
227/227 [==============================] - 289s 1s/step - loss: 0.0077 - accuracy: 0.9965 - val_loss: 0.0275 - val_accuracy: 0.9925
Epoch 28/50
227/227 [==============================] - 251s 1s/step - loss: 0.0073 - accuracy: 0.9980 - val_loss: 0.0222 - val_accuracy: 0.9938
Epoch 29/50
227/227 [==============================] - 244s 1s/step - loss: 0.0045 - accuracy: 0.9984 - val_loss: 0.0622 - val_accuracy: 0.9838
Epoch 30/50
227/227 [==============================] - 250s 1s/step - loss: 0.0083 - accuracy: 0.9955 - val_loss: 0.0300 - val_accuracy: 0.9913
Epoch 31/50
227/227 [==============================] - 222s 980ms/step - loss: 0.0040 - accuracy: 0.9988 - val_loss: 0.0206 - val_accuracy: 0.9938
Epoch 32/50
227/227 [==============================] - 223s 983ms/step - loss: 0.0191 - accuracy: 0.9941 - val_loss: 0.0235 - val_accuracy: 0.9925
Epoch 33/50
227/227 [==============================] - 222s 976ms/step - loss: 0.0128 - accuracy: 0.9969 - val_loss: 0.0780 - val_accuracy: 0.9801
Epoch 34/50
227/227 [==============================] - 221s 974ms/step - loss: 0.1188 - accuracy: 0.9835 - val_loss: 0.0376 - val_accuracy: 0.9900
Epoch 35/50
227/227 [==============================] - 222s 980ms/step - loss: 0.0062 - accuracy: 0.9978 - val_loss: 0.0291 - val_accuracy: 0.9950
Epoch 36/50
227/227 [==============================] - 220s 968ms/step - loss: 0.0056 - accuracy: 0.9982 - val_loss: 0.0252 - val_accuracy: 0.9950
Epoch 37/50
227/227 [==============================] - 244s 1s/step - loss: 0.0057 - accuracy: 0.9984 - val_loss: 0.1136 - val_accuracy: 0.9664
Epoch 38/50
227/227 [==============================] - 258s 1s/step - loss: 0.0343 - accuracy: 0.9871 - val_loss: 0.0334 - val_accuracy: 0.9900
Epoch 39/50
227/227 [==============================] - 256s 1s/step - loss: 0.0115 - accuracy: 0.9975 - val_loss: 0.0576 - val_accuracy: 0.9813
Epoch 40/50
227/227 [==============================] - 251s 1s/step - loss: 0.0187 - accuracy: 0.9915 - val_loss: 0.0638 - val_accuracy: 0.9888
Epoch 41/50
227/227 [==============================] - 269s 1s/step - loss: 0.0042 - accuracy: 0.9988 - val_loss: 0.0335 - val_accuracy: 0.9925
Epoch 42/50
227/227 [==============================] - 306s 1s/step - loss: 0.0048 - accuracy: 0.9990 - val_loss: 0.0441 - val_accuracy: 0.9900
Epoch 43/50
227/227 [==============================] - 282s 1s/step - loss: 0.0119 - accuracy: 0.9959 - val_loss: 0.0491 - val_accuracy: 0.9863
Epoch 44/50
227/227 [==============================] - 277s 1s/step - loss: 0.0111 - accuracy: 0.9963 - val_loss: 0.0317 - val_accuracy: 0.9888
Epoch 45/50
227/227 [==============================] - 234s 1s/step - loss: 0.0031 - accuracy: 0.9996 - val_loss: 0.0342 - val_accuracy: 0.9938
Epoch 46/50
227/227 [==============================] - 224s 985ms/step - loss: 0.0038 - accuracy: 0.9981 - val_loss: 0.0388 - val_accuracy: 0.9938
Epoch 47/50
227/227 [==============================] - 246s 1s/step - loss: 0.0030 - accuracy: 0.9990 - val_loss: 0.0336 - val_accuracy: 0.9925
Epoch 48/50
227/227 [==============================] - 254s 1s/step - loss: 0.0066 - accuracy: 0.9976 - val_loss: 0.0509 - val_accuracy: 0.9888
Epoch 49/50
227/227 [==============================] - 246s 1s/step - loss: 0.0042 - accuracy: 0.9984 - val_loss: 0.0424 - val_accuracy: 0.9938
Epoch 50/50
227/227 [==============================] - 243s 1s/step - loss: 0.0056 - accuracy: 0.9984 - val_loss: 0.0413 - val_accuracy: 0.9913
```

Out[34]:
```
<keras.callbacks.History at 0x2083eebcfa0>
```

Model evaluation :

In [35]:
```python
### Evaluating the model :
model.evaluate(X_test_scaled, y_test_arr)
```

```
1/1 [==============================] - 1s 1s/step - loss: 2.2368 - accuracy: 0.7500
```

Out[35]:
```
[2.236823797225952, 0.75]
```

In [50]:
```python
y_pred = model.predict(X_test_scaled)
y_pred[0]
```

Out[50]:
```
array([8.6779174e-08, 9.9999988e-01], dtype=float32)
```

In [51]:
```python
y_preds = []
for prediction in y_pred:
    y_preds.append(np.argmax(prediction))
```

In [54]:
```python
y_preds
```

Out[54]:
```
[1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1]
```

In [57]:
```python
confusion_matrix(y_preds, y_test_arr)
```

Out[57]:
```
array([[9, 4],
       [1, 6]], dtype=int64)
```

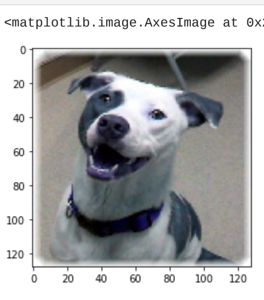
In [56]:
```python
print(classification_report(y_preds, y_test_arr))
```

```
              precision    recall  f1-score   support

           0       0.90      0.69      0.78        13
           1       0.60      0.86      0.71         7

    accuracy                           0.75        20
   macro avg       0.75      0.77      0.74        20
weighted avg       0.80      0.75      0.76        20
```

Prediction of a single image :

In [68]:
```python
### Image for prediction :
plt.imshow(X_test_scaled[0])
```

Out[68]:
```
<matplotlib.image.AxesImage at 0x20709fe75e0>
```



In [79]:
```python
### Predicting by giving image to the model :
prediction = model.predict(X_test_scaled[ : 1])
print(CATEGORIES[np.argmax(prediction)])
```

```
dogs
```

In [ ]: