Build a model to predict the category of an animal: dog or cat?

Input Dataset

The dataset is taken from kaggle dataset of image prediction of dogs and cats. https://www.kaggle.com/code/spiritedcoder/catdogclassification/input

The data is loaded from kaggle to google colab.

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!kaggle datasets download -d tongpython/cat-and-dog
import zipfile
zip_ref = zipfile.ZipFile('/content/cat-and-dog.zip','r')
zip_ref.extractall('/content')
zip_ref.close()
```

The code creates a directory named ".kaggle" in the home directory if it doesn't already exist. It then copies a file named "kaggle.json" to the ".kaggle" directory. Next, it downloads a dataset named "cat-and-dog" from Kaggle. After that, it imports the "zipfile" module. It creates a ZipFile object named "zip_ref" for the file "cat-and-dog.zip" in read mode. It extracts all the files from the zip file to the "/content" directory. Finally, it closes the ZipFile object.

Import Library

```
import pandas as pd
import numpy as np
import os
import tensorflow as tf
import keras
import sklearn
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.preprocessing import image
import matplotlib.pyplot as plt
```

All the required libraries are imported.

Image Preprocessing

```
image_size = 128
batch_size = 32
# Images preprocessing before training
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2, zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
training_set = train_datagen.flow_from_directory("/content/training_set/training_set",
                                                 target_size = (IMAGE_SIZE, IMAGE_SIZE),
                                                 batch_size = BATCH_SIZE,
                                                 class_mode = 'binary')

test_set = test_datagen.flow_from_directory("/content/test_set/test_set",
                                            target_size = (IMAGE_SIZE, IMAGE_SIZE),
                                            batch_size = BATCH_SIZE,
                                            class_mode = 'binary')
```

The code is setting up data generators for image preprocessing before training a machine learning model. The train_datagen is an ImageDataGenerator object that applies various transformations to the training images, including rescaling, shearing, zooming, and horizontal flipping. The test_datagen is another ImageDataGenerator object that only applies rescaling to the test images. The training_set is a generator that loads the training images from a specified directory (/content/training_set/training_set), resizes them to a target size of 128x128 pixels, and creates batches of 32 images at a time. The class mode is set to 'binary', indicating that the images are classified into two categories. Similarly, the test_set is a generator that loads the test images from a specified directory (/content/test_set/test_set), resizes them to the same target size, and creates batches of 32 images. The class mode is also set to 'binary'.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3),
activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(units= 64, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(units = 1, activation = 'sigmoid'))
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
```

The code defines a convolutional neural network (CNN) model using the Keras library.
The model consists of multiple layers:

The first layer is a convolutional layer with 32 filters of size 3x3, using the ReLU activation function. The input shape is specified as (IMAGE_SIZE, IMAGE_SIZE, 3), indicating an image with RGB channels.
The second layer is a max pooling layer with a pool size of 2x2, which reduces the spatial dimensions of the input by taking the maximum value in each 2x2 window.
The third layer is another convolutional layer with 32 filters of size 3x3, using the ReLU activation function.
The fourth layer is another max pooling layer with a pool size of 2x2.
The fifth layer is a third convolutional layer with 32 filters of size 3x3, using the ReLU activation function.
The sixth layer is a third max pooling layer with a pool size of 2x2.
The seventh layer is a flatten layer, which converts the multidimensional output of the previous layer into a one-dimensional vector.
The eighth layer is a dense (fully connected) layer with 64 units, using the ReLU activation function.
The ninth layer is a dropout layer, which randomly sets a fraction of input units to 0 at each update during training to prevent overfitting.
The tenth layer is another dense layer with 1 unit, using the sigmoid activation function. This is the final output layer of the model.
After defining the model architecture, the code compiles the model using the Adam optimizer, binary cross-entropy as the loss function (since it is a binary classification problem), and accuracy as the metric to monitor during training.

```
model.fit(x = training_set, validation_data = test_set, epochs = 5)
```

The code is fitting a machine learning model using the training set and validating it using the test set for 5 epochs.

```
print("The model class indices are: ", training_set.class_indices)
```

The code prints the message "The model class indices are: " followed by the class indices of the training set.

```
# testing on a single input
test_image = image.load_img("/content/test_set/test_set/dogs/dog.4088.jpg", target_size = (IMAGE_SIZE, IMAGE_SIZE))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
print(test_image)
result = model.predict(test_image)
if result[0][0] == 1:
   prediction = 'dog'
else:
   prediction = 'cat'
```

This code loads an image, resizes it to a specified target size, converts it to an array, and expands its dimensions. It then uses a trained model to predict whether the image contains a dog or a cat, and assigns the corresponding label to the variable "prediction".

```
print(prediction)
```

The given code prints the value of the variable "prediction" to the console or standard output which is the predicted animal cat or a dog.
Thus model is built which predicts the category of an animal from cat and dog.