```
# Import necessary libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
import random
from keras.preprocessing.image import load_img
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
from keras import Sequential
from keras.layers import Conv2D, MaxPool2D, Flatten, Dense

warnings.filterwarnings('ignore')


# download and unzip the file consisting the dataset
!wget https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_5340.zip
!unzip kagglecatsanddogs_5340.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: PetImages/Dog/5500.jpg
  inflating: PetImages/Dog/5501.jpg
  inflating: PetImages/Dog/5502.jpg
  inflating: PetImages/Dog/5503.jpg
  inflating: PetImages/Dog/5504.jpg
  inflating: PetImages/Dog/5505.jpg
  inflating: PetImages/Dog/5506.jpg
  inflating: PetImages/Dog/5507.jpg
  inflating: PetImages/Dog/5508.jpg
  inflating: PetImages/Dog/5509.jpg
  inflating: PetImages/Dog/551.jpg
  inflating: PetImages/Dog/5510.jpg
  inflating: PetImages/Dog/5511.jpg
  inflating: PetImages/Dog/5512.jpg
  inflating: PetImages/Dog/5513.jpg
  inflating: PetImages/Dog/5514.jpg
  inflating: PetImages/Dog/5515.jpg
  inflating: PetImages/Dog/5516.jpg
  inflating: PetImages/Dog/5517.jpg
  inflating: PetImages/Dog/5518.jpg
  inflating: PetImages/Dog/5519.jpg
  inflating: PetImages/Dog/552.jpg
  inflating: PetImages/Dog/5520.jpg
  inflating: PetImages/Dog/5521.jpg
  inflating: PetImages/Dog/5522.jpg
  inflating: PetImages/Dog/5523.jpg
  inflating: PetImages/Dog/5524.jpg
  inflating: PetImages/Dog/5525.jpg
  inflating: PetImages/Dog/5526.jpg
  inflating: PetImages/Dog/5527.jpg
  inflating: PetImages/Dog/5528.jpg
  inflating: PetImages/Dog/5529.jpg
  inflating: PetImages/Dog/553.jpg
  inflating: PetImages/Dog/5530.jpg
  inflating: PetImages/Dog/5531.jpg
  inflating: PetImages/Dog/5532.jpg
  inflating: PetImages/Dog/5533.jpg
  inflating: PetImages/Dog/5534.jpg
  inflating: PetImages/Dog/5535.jpg
  inflating: PetImages/Dog/5536.jpg
  inflating: PetImages/Dog/5537.jpg
  inflating: PetImages/Dog/5538.jpg
  inflating: PetImages/Dog/5539.jpg
  inflating: PetImages/Dog/554.jpg
  inflating: PetImages/Dog/5540.jpg
  inflating: PetImages/Dog/5541.jpg
  inflating: PetImages/Dog/5542.jpg
  inflating: PetImages/Dog/5543.jpg
  inflating: PetImages/Dog/5544.jpg
  inflating: PetImages/Dog/5545.jpg
  inflating: PetImages/Dog/5546.jpg
  inflating: PetImages/Dog/5547.jpg
  inflating: PetImages/Dog/5548.jpg
  inflating: PetImages/Dog/5549.jpg
  inflating: PetImages/Dog/555.jpg
  inflating: PetImages/Dog/5550.jpg
  inflating: PetImages/Dog/5551.jpg
```

```
# Initialize lists to store image paths and labels
```

```
input_path = []
label = []

for class_name in os.listdir("PetImages"):
    for path in os.listdir("PetImages/"+class_name):
        if class_name == 'Cat':
            label.append(0)
        else:
            label.append(1)
        input_path.append(os.path.join("PetImages", class_name, path))


# Removing invalid image paths

invalid_images = ['PetImages/Cat/666.jpg', 'PetImages/Cat/Thumbs.db', 'PetImages/Dog/Thumbs.db', 'PetImages/Dog/11702.jpg']
input_path = [path for path in input_path if path not in invalid_images]


# Initialize lists to store image paths and labels

input_path = []
label = []

# Iterate through the subdirectories in the image directory and append image paths along with their labels to input_path and label lists res
for class_name in os.listdir("PetImages"):
    for path in os.listdir("PetImages/"+class_name):
        if class_name == 'Cat':
            label.append(0)
        else:
            label.append(1)
        input_path.append(os.path.join("PetImages", class_name, path))
print(input_path[0], label[0])
```

```
PetImages/Dog/9347.jpg 1
```

```
# Create a DataFrame with image paths and labels, shuffle the DataFrame, and reset index.

df = pd.DataFrame()
df['images'] = input_path
df['label'] = label
df = df.sample(frac=1).reset_index(drop=True)
df.head()
```

	images	label
0	PetImages/Cat/7257.jpg	0
1	PetImages/Cat/3735.jpg	0
2	PetImages/Cat/10125.jpg	0
3	PetImages/Dog/9942.jpg	1
4	PetImages/Cat/10157.jpg	0

```
# Checking for image paths that do not end with '.jpg' and printing them.

for i in df['images']:
    if '.jpg' not in i:
        print(i)
```

```
PetImages/Cat/Thumbs.db
PetImages/Dog/Thumbs.db
```

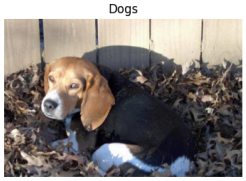
```
# Attempt to open each image using PIL.Image.open() and store paths of images that raise exceptions in a list.

import PIL
l = []
for image in df['images']:
    try:
        img = PIL.Image.open(image)
    except:
        l.append(image)
l
```

```
['PetImages/Cat/Thumbs.db',
 'PetImages/Dog/Thumbs.db',
 'PetImages/Cat/666.jpg',
 'PetImages/Dog/11702.jpg']
```

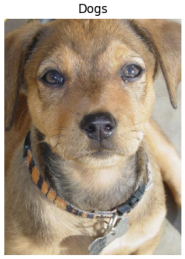
```
# Filtering out specific image paths from the DataFrame where images are considered invalid, and getting the length of the DataFrame.


df = df[df['images']!='PetImages/Dog/Thumbs.db']
df = df[df['images']!='PetImages/Cat/Thumbs.db']
df = df[df['images']!='PetImages/Cat/666.jpg']
df = df[df['images']!='PetImages/Dog/11702.jpg']
len(df)
```

```
24998
```

```
# Adding import statement

import matplotlib.pyplot as plt  # Add this import statement



# Plot a grid of 25 randomly selected dog images from the DataFrame.

plt.figure(figsize=(25,25))
temp = df[df['label']==1]['images']
start = random.randint(0, len(temp))
files = temp[start:start+25]

for index, file in enumerate(files):
    plt.subplot(5,5, index+1)
    img = load_img(file)
    img = np.array(img)
    plt.imshow(img)
    plt.title('Dogs')
    plt.axis('off')
```

Dogs 

Dogs 

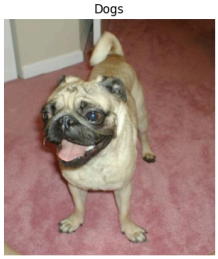
Dogs 

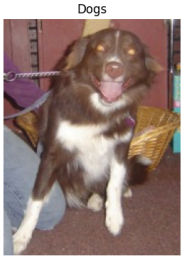
Dogs 

Dogs 

Dogs 

Dogs 

Dogs 

Dogs 

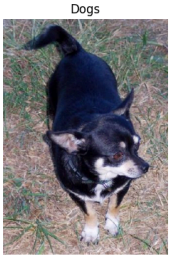
Dogs 

Dogs 

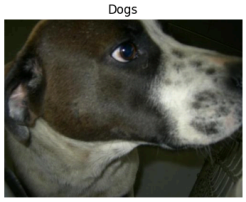
Dogs 

Dogs 

Dogs 

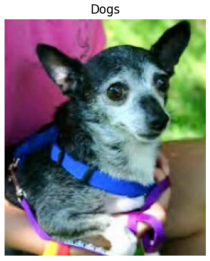
Dogs 

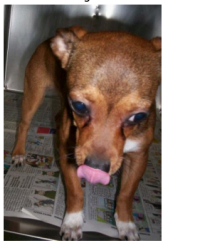
Dogs 

Dogs 

Dogs 

Dogs 

Dogs 

Dogs 

Dogs 

Dogs 

Dogs 

Dogs 

Dogs

Dogs

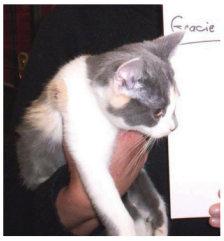
Dogs

Dogs

Dogs

```
# Plot a grid of 25 randomly selected cat images from the DataFrame.

plt.figure(figsize=(25,25))
temp = df[df['label']==0]['images']
start = random.randint(0, len(temp))
files = temp[start:start+25]

for index, file in enumerate(files):
    plt.subplot(5,5, index+1)
    img = load_img(file)
    img = np.array(img)
    plt.imshow(img)
    plt.title('Cats')
    plt.axis('off')
```

Cats 

Cats 

Cats 

Cats 

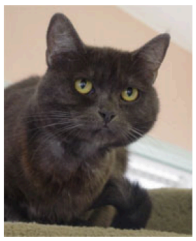
Cats 

Cats 

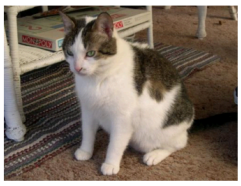
Cats 

Cats 

Cats 

Cats 

Cats 

Cats 

Cats 

Cats 

Cats 

Cats 

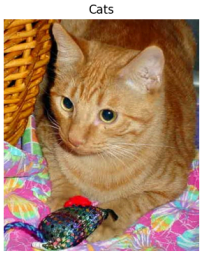
Cats 

Cats 

Cats 

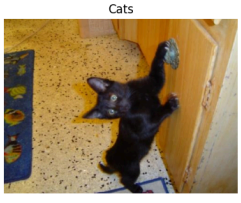
Cats 

Cats 

Cats 

Cats 

Cats 

Cats 

Cats

Cats

Cats

Cats

Cats

```
# Split the DataFrame into training and testing sets

train, test = train_test_split(df, test_size=0.2, random_state=42)



# Creating image data generators for training and validation sets with specified augmentation parameters.


train_generator = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 40,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True,
    fill_mode = 'nearest'
)

val_generator = ImageDataGenerator(rescale = 1./255)



# Converting the label column in the DataFrame to string data type.


df['label'] = df['label'].astype('str')




# Displaying the first few rows of the DataFrame.


df.head()
```

	images	label
0	PetImages/Cat/7257.jpg	0
1	PetImages/Cat/3735.jpg	0
2	PetImages/Cat/10125.jpg	0
3	PetImages/Dog/9942.jpg	1
4	PetImages/Cat/10157.jpg	0

```
#Additional Step: Experiment with Different Architectures
# Define a function to create different model architectures

def create_model(input_shape):
    model = Sequential([
        Conv2D(32, (3,3), activation='relu', input_shape=input_shape),
        MaxPool2D((2,2)),
        Conv2D(64, (3,3), activation='relu'),
        MaxPool2D((2,2)),
        Conv2D(128, (3,3), activation='relu'),
        MaxPool2D((2,2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model


# Hyperparameter Tuning
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size=0.2, random_state=42)


# Defining ImageDataGenerator objects for training and validation data, performing normalization and image augmentation.

from keras.preprocessing.image import ImageDataGenerator
train_generator = ImageDataGenerator(
    rescale = 1./255,  # normalization of images
    rotation_range = 40, # augmention of images to avoid overfitting
    shear_range = 0.2,
    zoom_range = 0.2,
```

```
    horizontal_flip = True,
    fill_mode = 'nearest'
)

val_generator = ImageDataGenerator(rescale = 1./255)

# Generate batches of augmented data for training using the flow_from_dataframe method.
train_iterator = train_generator.flow_from_dataframe(
    train,
    x_col='images',
    y_col='label',
    target_size=(128,128),
    batch_size=512,
    class_mode='binary'
)
```

```
Found 19998 validated image filenames belonging to 2 classes.
```

```
# Generating batches of augmented data for validation using the flow_from_dataframe method.

val_iterator = val_generator.flow_from_dataframe(
    test,
    x_col='images',
    y_col='label',
    target_size=(128,128),
    batch_size=512,
    class_mode='binary'
)
```

```
Found 5000 validated image filenames belonging to 2 classes.
```

```
# Defining the convolutional neural network (CNN) model architecture using Keras Sequential API.


from keras import Sequential
from keras.layers import Conv2D, MaxPool2D, Flatten, Dense

model = Sequential([
                    Conv2D(16, (3,3), activation='relu', input_shape=(128,128,3)),
                    MaxPool2D((2,2)),
                    Conv2D(32, (3,3), activation='relu'),
                    MaxPool2D((2,2)),
                    Conv2D(64, (3,3), activation='relu'),
                    MaxPool2D((2,2)),
                    Flatten(),
                    Dense(512, activation='relu'),
                    Dense(1, activation='sigmoid')
])



# Compiling the CNN model with Adam optimizer, binary crossentropy loss function, and accuracy metric.
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Printing the summary of the model architecture.
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 16)      448

 max_pooling2d (MaxPooling2  (None, 63, 63, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 61, 61, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 30, 30, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 28, 28, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)
```