Cat & Dog Classification using Convolutional Neural Network in Python

Last Updated : 09 Sep, 2024

Image Classification is one of the most interesting and useful applications of Deep neural networks and Convolutional Neural Networks that enables us to automate the task of assembling similar images and arranging data without the supervision of real humans.

Cat & Dog Classification using Convolutional Neural Network in Python

In this article, we will learn how to build a classifier using a simple Convolution Neural Network which can classify the images of dogs and cats.

To get more understanding, follow the steps accordingly.

Importing Libraries

The libraries we will using are :

- Pandas – This library is used to load 2D array format and DataFrames.
- Numpy – It is used to perform large computations in a very short time.
- Matplotlib – This library is used to draw visualizations.
- Sklearn – This module contains multiple libraries having pre-implemented functions to perform tasks from data preprocessing to model development and evaluation.

- OpenCV – This is an open-source library mainly focused on image processing and handling.
- Tensorflow – This is an open-source library that is used for Machine Learning and Artificial intelligence and provides a range of functions to achieve complex functionalities with single lines of code.

```
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
import numpy as np

import warnings
warnings.filterwarnings('ignore')

from tensorflow import keras
from keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.utils import image_dataset_from_directory
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras.preprocessing import image_dataset_from_directory

import os
import matplotlib.image as mpimg
```

Importing Dataset

The dataset is in the format of a zip file containing 2 folders : Cat and Dog. Further each folder contains 12500 images of respective animals.

So to import and then unzip it, you can run the below code.

```
from zipfile import ZipFile

data_path = 'dog-vs-cat.zip'

with ZipFile(data_path, 'r') as zip:
    zip.extractall()
    print('The data set has been extracted.')
```

Data Visualization

In this section, we will try to understand visualize some images which have been provided to us to build the classifier for each class.

```
path = 'dog-vs-cat'
classes = os.listdir(path)
classes
```

```
['cats', 'dogs']
```

This shows that, there are two classes that we have here i.e. Cat and Dog.

```
fig = plt.gcf()
fig.set_size_inches(16, 16)

cat_dir = os.path.join('dog-vs-cat-classification/cat')
dog_dir = os.path.join('dog-vs-cat-classification/dog')
cat_names = os.listdir(cat_dir)
dog_names = os.listdir(dog_dir)

pic_index = 210

cat_images = [os.path.join(cat_dir, fname)
              for fname in cat_names[pic_index-8:pic_index]]
dog_images = [os.path.join(dog_dir, fname)
              for fname in dog_names[pic_index-8:pic_index]]

for i, img_path in enumerate(cat_images + dog_images):
    sp = plt.subplot(4, 4, i+1)
    sp.axis('Off')

    img = mpimg.imread(img_path)
    plt.imshow(img)

plt.show()
```

Output :

Data Preparation for Training

In this section, we will classify the dataset into train and validation format.

```
base_dir = 'dog-vs-cat-classification'

# Create datasets
train_datagen = image_dataset_from_directory(base_dir,
                                                  image_size=(200,200),
                                                  subset='training',
                                                  seed = 1,
                                                 validation_split=0.1,
                                                  batch_size= 32)
test_datagen = image_dataset_from_directory(base_dir,
                                                  image_size=(200,200),
                                                  subset='validation',
                                                  seed = 1,
                                                 validation_split=0.1,
                                                  batch_size= 32)
```

Output :

```
Found 25000 files belonging to 2 classes.
Using 22500 files for training.
Found 25000 files belonging to 2 classes.
Using 2500 files for validation.
```

Model Architecture

The model will contain the following Layers:

- Four Convolutional Layers followed by MaxPooling Layers.
- The Flatten layer to flatten the output of the convolutional layer.
- Then we will have three fully connected layers followed by the output of the flattened layer.
- We have included some BatchNormalization layers to enable stable and fast training and a Dropout layer before the final layer to avoid any possibility of overfitting.
- The final layer is the output layer which has the activation function sigmoid to classify the results into two classes.

```
model = tf.keras.models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(200, 200, 3)),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.1),
    layers.BatchNormalization(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.2),
    layers.BatchNormalization(),
    layers.Dense(1, activation='sigmoid')
])
```

Let's print the summary of the model's architecture:

```
model.summary()
```

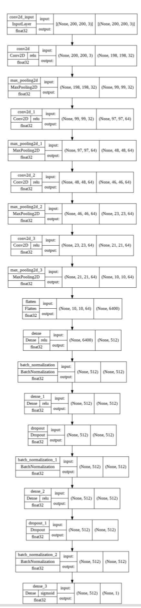
Output :

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 198, 198, 32)      896

 max_pooling2d (MaxPooling2D  (None, 99, 99, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 97, 97, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 48, 48, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 46, 46, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 23, 23, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 21, 21, 64)        36928

 max_pooling2d_3 (MaxPooling  (None, 10, 10, 64)       0
 2D)

 flatten (Flatten)           (None, 6400)              0

 dense (Dense)               (None, 512)               3277312

 batch_normalization (BatchN  (None, 512)              2048
 ormalization)

 dense_1 (Dense)             (None, 512)               262656

 dropout (Dropout)           (None, 512)               0

 batch_normalization_1 (Batc  (None, 512)              2048
 hNormalization)

 dense_2 (Dense)             (None, 512)               262656

 dropout_1 (Dropout)         (None, 512)               0

 batch_normalization_2 (Batc  (None, 512)              2048
 hNormalization)

 dense_3 (Dense)             (None, 1)                 513
_________________________________________________________________
```

The input image we have taken initially resized into 200 X 200. And later it transformed into the binary classification value. To understand the huge number of parameters and complexity of the model which helps us to achieve a high-performance model let's see the plot_model.

```
keras.utils.plot_model(
    model,
    show_shapes=True,
    show_dtype=True,
    show_layer_activations=True
)
```

Output :



```
model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
```

```
    metrics=['accuracy']
)
```

Model Training

Now we will train our model, the model is working fine on epochs = 10, but you can perform hyperparameter tuning for better results.

```
history = model.fit(train_datagen,
          epochs=10,
          validation_data=test_datagen)
```

Output :

```
Epoch 1/10
704/704 [==============================] - 56s 60ms/step - loss: 0.6837 - accuracy: 0.6174 - val_loss: 0.6314 - val_accuracy: 0.6824
Epoch 2/10
704/704 [==============================] - 42s 59ms/step - loss: 0.5959 - accuracy: 0.6756 - val_loss: 0.7091 - val_accuracy: 0.5256
Epoch 3/10
704/704 [==============================] - 42s 59ms/step - loss: 0.5385 - accuracy: 0.7295 - val_loss: 1.3551 - val_accuracy: 0.5240
Epoch 4/10
704/704 [==============================] - 43s 60ms/step - loss: 0.4555 - accuracy: 0.7871 - val_loss: 0.4511 - val_accuracy: 0.7908
Epoch 5/10
704/704 [==============================] - 43s 60ms/step - loss: 0.4637 - accuracy: 0.7799 - val_loss: 0.7524 - val_accuracy: 0.6628
Epoch 6/10
704/704 [==============================] - 42s 59ms/step - loss: 0.3903 - accuracy: 0.8249 - val_loss: 0.6068 - val_accuracy: 0.7432
Epoch 7/10
704/704 [==============================] - 41s 58ms/step - loss: 0.3705 - accuracy: 0.8323 - val_loss: 0.6982 - val_accuracy: 0.7012
Epoch 8/10
704/704 [==============================] - 42s 59ms/step - loss: 0.3206 - accuracy: 0.8611 - val_loss: 0.3754 - val_accuracy: 0.8360
Epoch 9/10
```

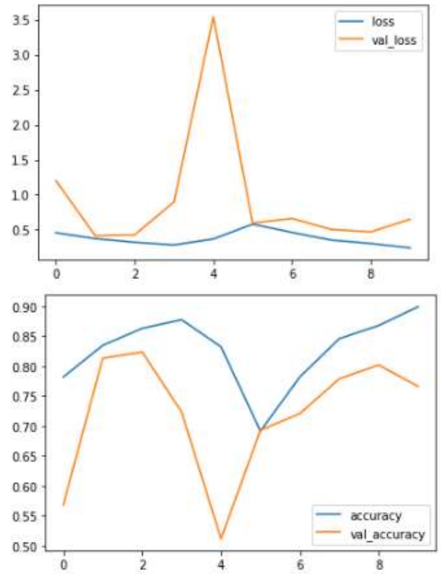
Model Training

Model Evaluation

Let's visualize the training and validation accuracy with each epoch.

```
history_df = pd.DataFrame(history.history)
history_df.loc[:, ['loss', 'val_loss']].plot()
history_df.loc[:, ['accuracy', 'val_accuracy']].plot()
plt.show()
```

Output :



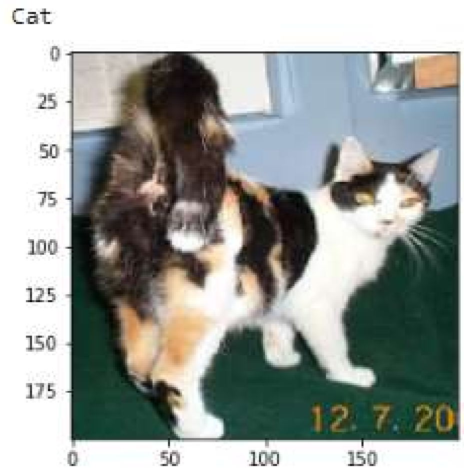
training and validation accuracy

Model Testing and Prediction

Let’s check the model for random images.

```
from keras.preprocessing import image

#Input image
test_image = image.load_img('1.jpg',target_size=(200,200))

#For show image
plt.imshow(test_image)
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image,axis=0)

# Result array
result = model.predict(test_image)

#Mapping result array with the main name list
i=0
if(result>=0.5):
  print("Dog")
else:
  print("Cat")
```

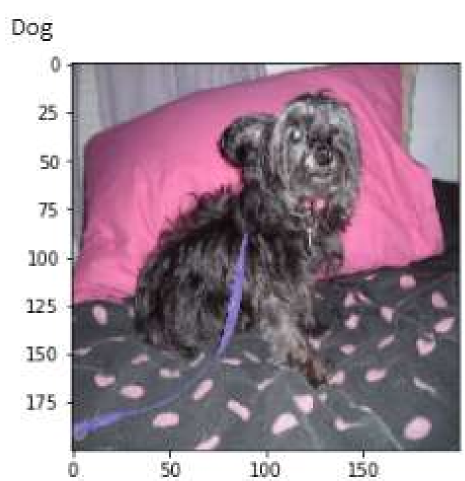
Output :



cat

```
test_image = image.load_img('test/2.jpg', target_size=(200, 200))

# For show image
plt.imshow(test_image)
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)

# Result array
result = model.predict(test_image)
# Mapping result array with the main name list
i = 0
if(result >= 0.5):
    print("Dog")
else:
    print("Cat")
```

Output:



dog

Get the complete notebook link here: