Classify images Cat vs Dog

Introduction

In this notebook, we will learn how to classify images of dog and cat by building a simple Neural Network.

Importing Libraries

In [2]:
```
import numpy as np
import sys
import tensorflow as tf
import os
import sys
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator


%matplotlib inline
import matplotlib.image as img
import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix
import plotly.graph_objects as go
import itertools
import plotly.express as px
```

In [3]:
```
seed = 0
np.random.seed(seed)
tf.random.set_seed(3)
```

Setting Path and Extracting Files

In [4]:
```
import zipfile

zip_files = ['test1', 'train']

for zip_file in zip_files:
    with zipfile.ZipFile("../input/dogs-vs-cats/{}.zip".format(zip_file),"r") as z:
        z.extractall(".")
        print("{} unzipped".format(zip_file))
```

```
test1 unzipped
train unzipped
```

In [5]:
```
'''test1, train Data is in current working folder'''
print(os.listdir('../working'))
```

```
['.virtual_documents', 'train', 'test1']
```

Importing Data

In [6]:
```
IMAGE_FOLDER_PATH = "../working/train"
FILE_NAMES = os.listdir(IMAGE_FOLDER_PATH)
WIDTH = 150
HEIGHT = 150
```

In [7]:
```
FILE_NAMES[0:5]
```

Out[7]:
```
['cat.4338.jpg',
 'cat.8662.jpg',
 'dog.10253.jpg',
 'dog.4451.jpg',
 'cat.3180.jpg']
```

In [8]:
```
labels = []
for i in os.listdir(IMAGE_FOLDER_PATH):
    labels+=[i]
```

Data distribuation of all data

In [9]:
```
# empty list
targets = list()
full_paths = list()
train_cats_dir = list()
train_dogs_dir = list()

# finding each file's target
for file_name in FILE_NAMES:
    target = file_name.split(".")[0] # target name
    full_path = os.path.join(IMAGE_FOLDER_PATH, file_name)

    if(target == "dog"):
        train_dogs_dir.append(full_path)
    if(target == "cat"):
        train_cats_dir.append(full_path)

    full_paths.append(full_path)
    targets.append(target)

dataset = pd.DataFrame() # make dataframe
dataset['image_path'] = full_paths # file path
dataset['target'] = targets # file's target
```

In [10]:
```
dataset.head(10)
```

Out[10]:

	image_path	target
0	../working/train/cat.4338.jpg	cat
1	../working/train/cat.8662.jpg	cat
2	../working/train/dog.10253.jpg	dog
3	../working/train/dog.4451.jpg	dog
4	../working/train/cat.3180.jpg	cat
5	../working/train/cat.9004.jpg	cat
6	../working/train/dog.3946.jpg	dog
7	../working/train/dog.3826.jpg	dog
8	../working/train/dog.8053.jpg	dog
9	../working/train/dog.8415.jpg	dog

In [11]:
```
print("total data counts:", dataset['target'].count())
counts = dataset['target'].value_counts()
print(counts)
```

```
total data counts: 25000
cat    12500
dog    12500
Name: target, dtype: int64
```

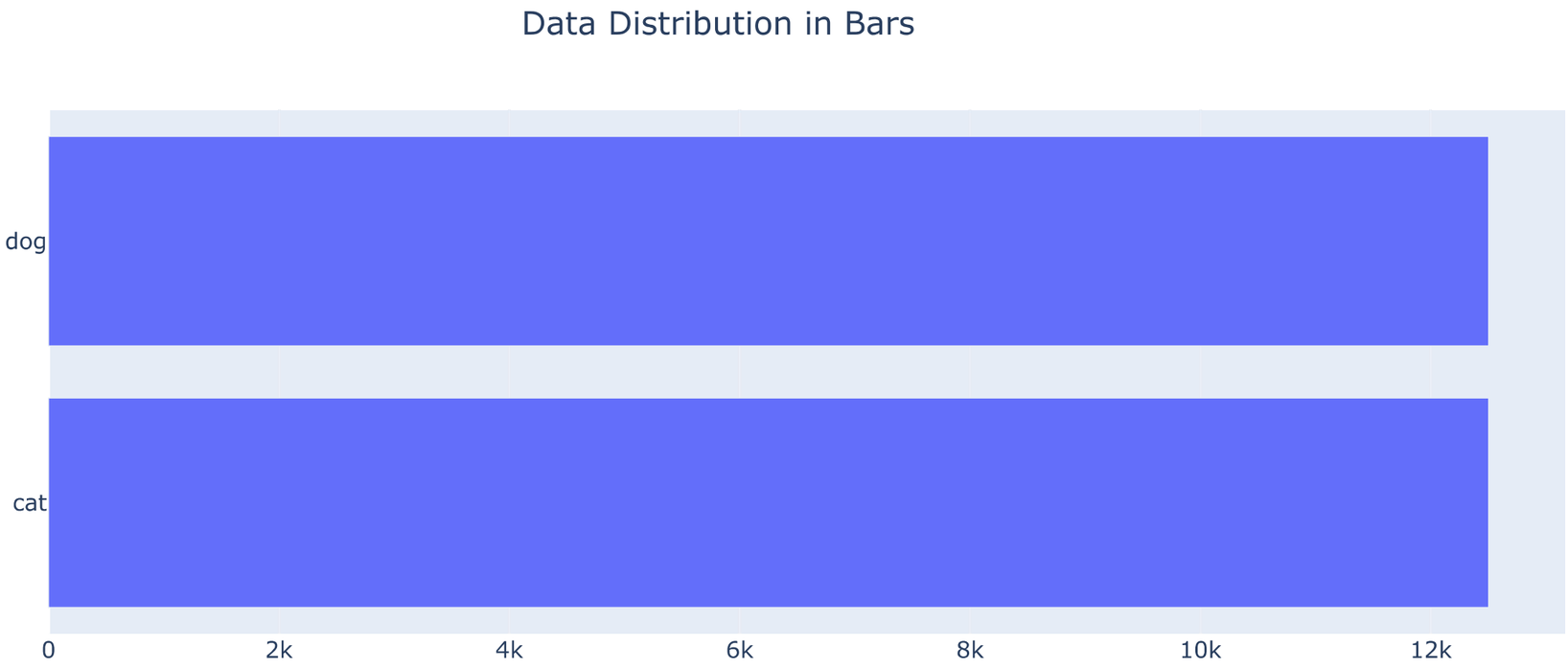
Below there are two chart

Bar Chart.

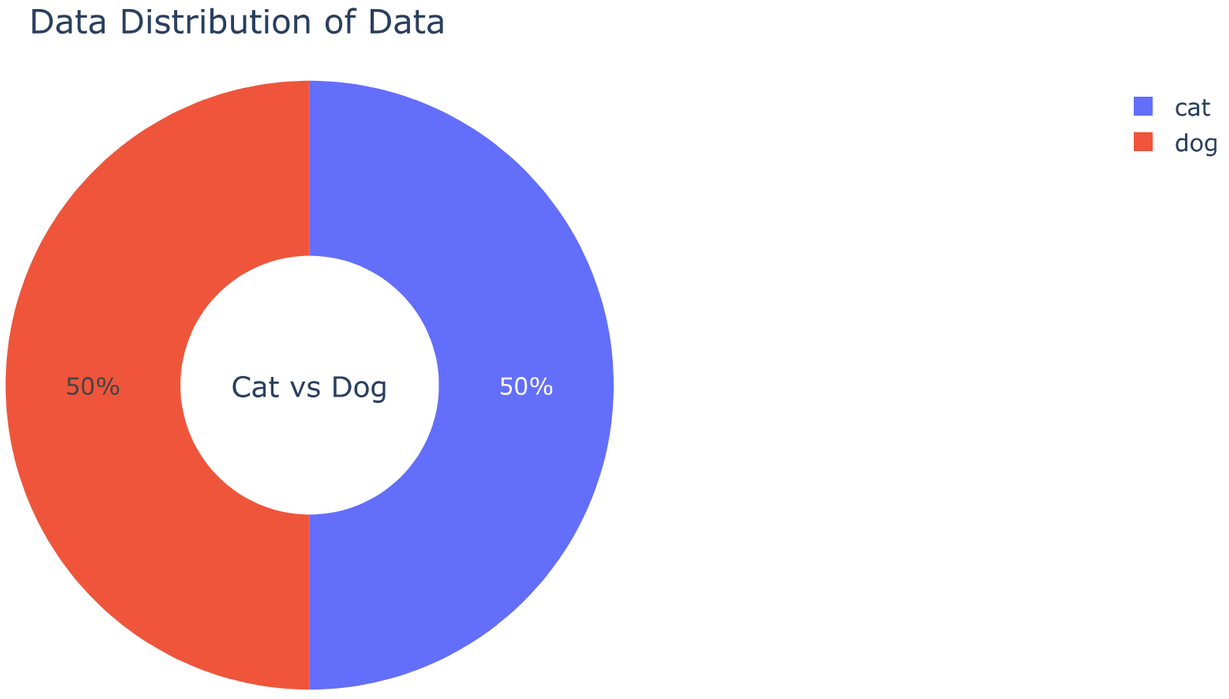
Pie Chart.

Data is equally distributed.

In [12]:
```
fig = go.Figure(go.Bar(
            x= counts.values,
            y=counts.index,
            orientation='h'))

fig.update_layout(title='Data Distribution in Bars',font_size=15,title_x=0.45)


fig.show()
```

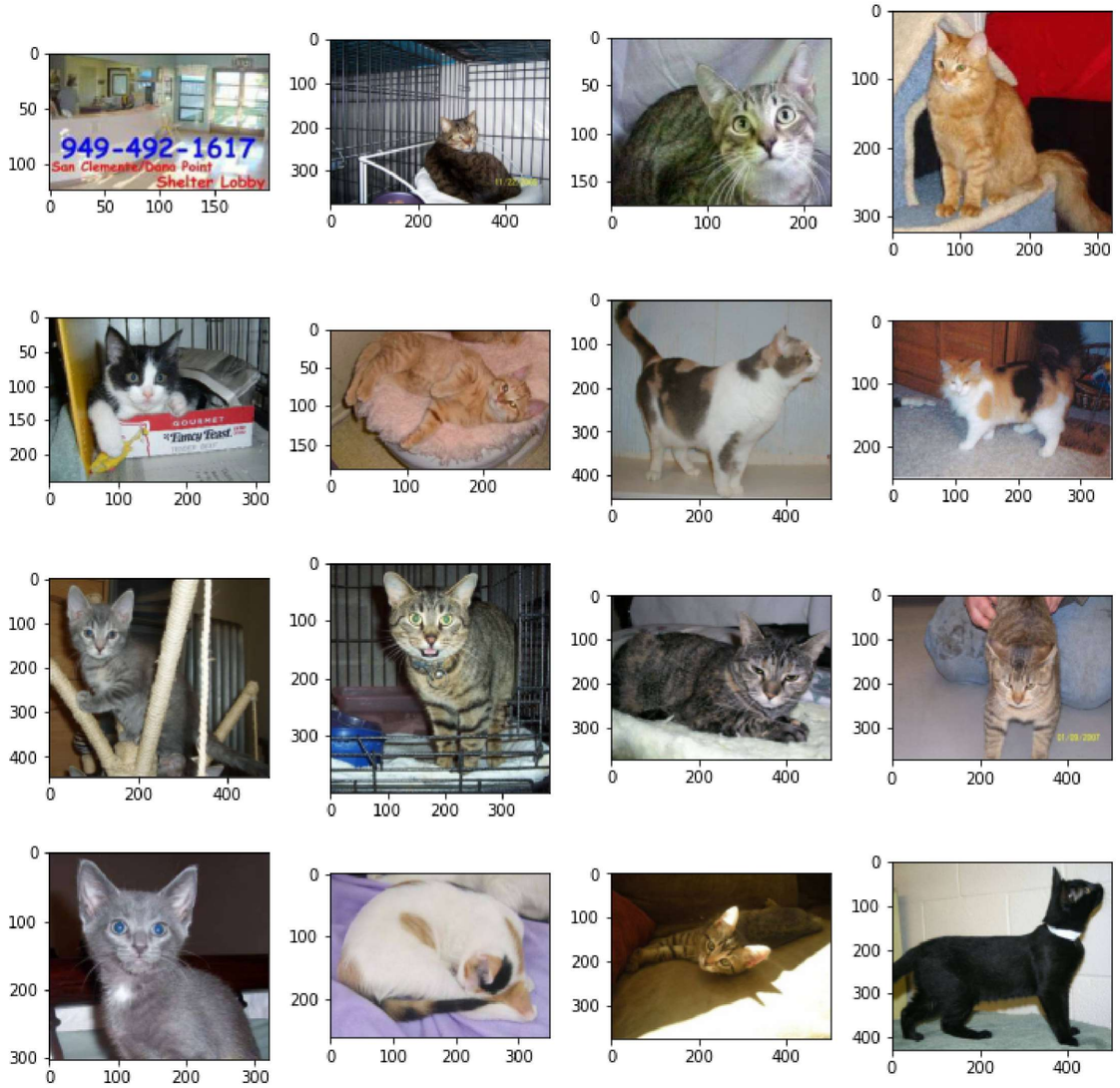


In [13]:
```
fig=px.pie(counts.head(10),values= 'target', names=dataset['target'].unique(),hole=0.425)
fig.update_layout(title='Data Distribution of Data',font_size=15,title_x=0.45,annotations=[dict(text='Cat vs Dog',font_size=18, showarrow=False,height=800,width=700)])
fig.update_traces(textfont_size=15,textinfo='percent')
fig.show()
```



Displaying Images of Cat

In [14]:
```
rows = 4
cols = 4
axes = []
fig=plt.figure(figsize=(10,10))
i = 0

for a in range(rows*cols):
    b = img.imread(train_cats_dir[i])
    axes.append(fig.add_subplot(rows,cols,a+1))
    plt.imshow(b)
    i+=1
fig.tight_layout()
plt.show()
```



Displaying Images of Dog

In [15]:
```
rows = 4
cols = 4
axes = []
fig=plt.figure(figsize=(10,10))
i = 0

for a in range(rows*cols):
    b = img.imread(train_dogs_dir[i])
    axes.append(fig.add_subplot(rows,cols,a+1))
    plt.imshow(b)
    i+=1
fig.tight_layout()
plt.show()
```

Siplitting the data

In [16]:
```
dataset_train, dataset_test = train_test_split(dataset, test_size=0.2, random_state=seed)
```

Data Distribution of Train Data

Below are Two Charts of Training Data Set

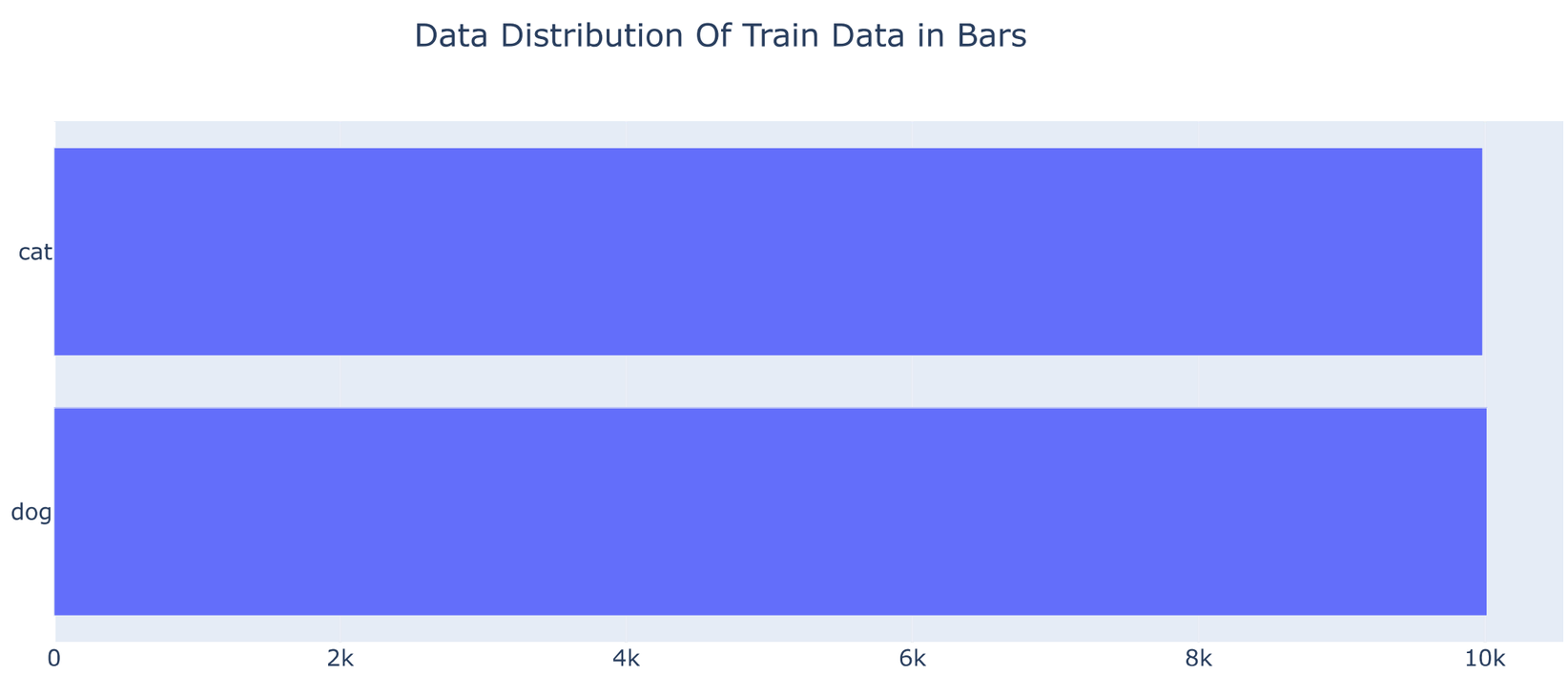
Bar Chart.

Pie Chart

In [17]:
```
class_id_distributionTrain = dataset_train['target'].value_counts()
class_id_distributionTrain.head(10)
```

Out[17]:
```
dog    10015
cat     9985
Name: target, dtype: int64
```

In [18]:
```
fig = go.Figure(go.Bar(
            x=class_id_distributionTrain.values,
            y=class_id_distributionTrain.index,
            orientation='h'))

fig.update_layout(title='Data Distribution Of Train Data in Bars',font_size=15,title_x=0.45)

fig.show()
```



In [19]:
```
fig=px.pie(class_id_distributionTrain.head(10),values= 'target', names=dataset_train['target'].unique(),hole=0.425)
fig.update_layout(title='Data Distribution of Train Data in Pie Chart',font_size=15,title_x=0.45,annotations=[dict(text='Cat vs Dog',font_size=18, showarrow=False,height=800,width=700)])
fig.update_traces(textfont_size=15,textinfo='percent')
fig.show()
```

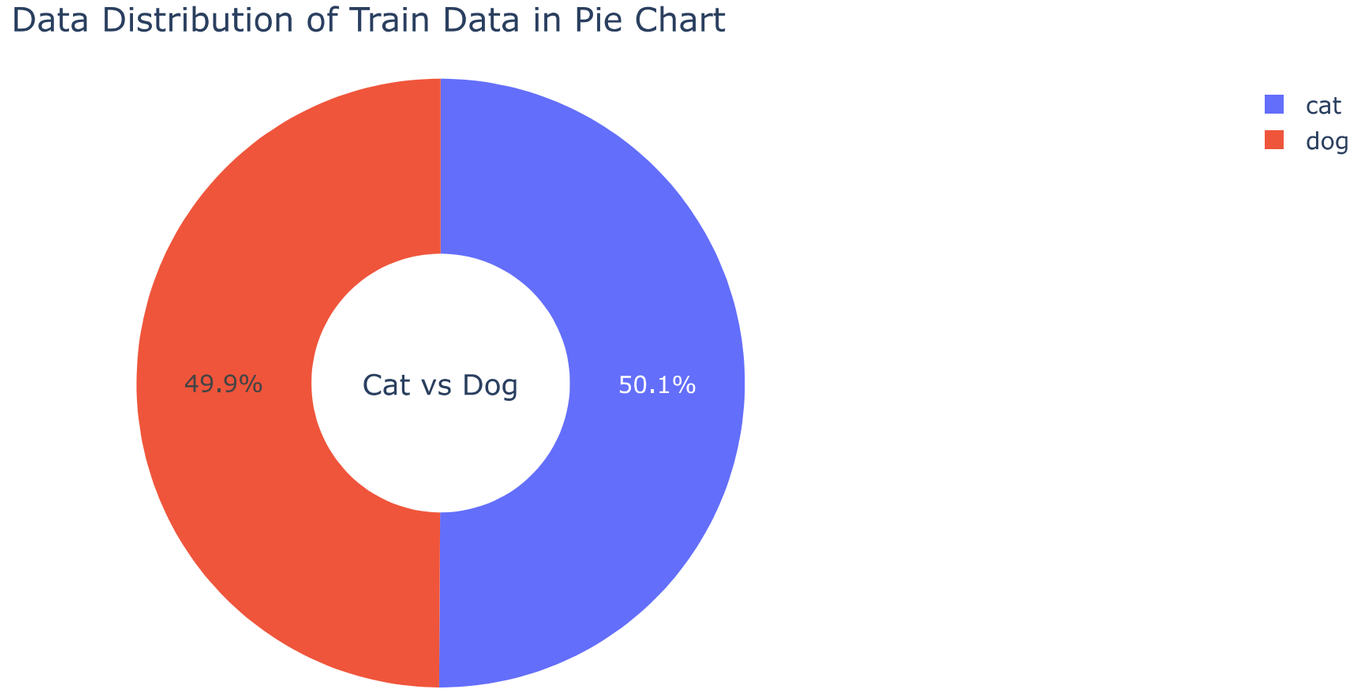


Image Data Generator

When there is little data to train, we have to use ImageDataGenerator to increase the number of data.

rescale = 1./255: This scales the pixel values between 0 and 1.

rotation_range = 15: Applies random rotation within a range of 15 degrees.

shear_range = 0.1: Introduces a shear range of 10%.

zoom_range = 0.2: Allows zooming within a range of 20%.

horizontal_flip = True: Enables random horizontal flipping.

width_shift_range = 0.1: Randomly shifts the original image horizontally within 10% of its width.

height_shift_range = 0.1: Randomly shifts the original image vertically within 10% of its height.

In [23]:
```
train_datagen=ImageDataGenerator(
rotation_range=15,
rescale=1./255,
shear_range=0.1,
zoom_range=0.2,
```

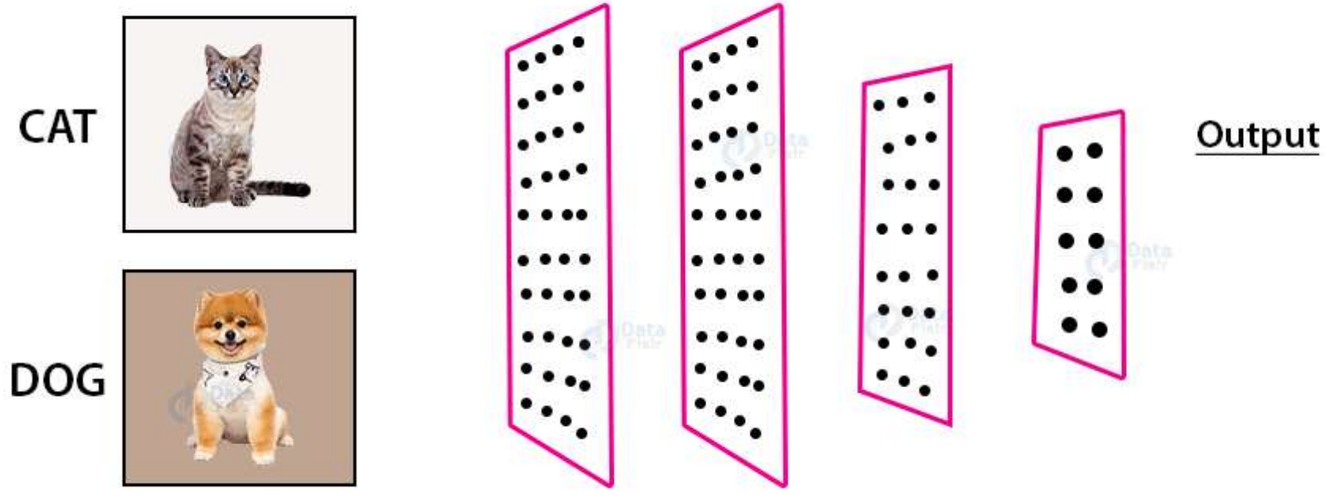
```
horizontal_flip=True,
width_shift_range=0.1,
height_shift_range=0.1)

train_datagenerator=train_datagen.flow_from_dataframe(dataframe=dataset_train,
                                                      x_col="image_path",
                                                      y_col="target",
                                                      target_size=(WIDTH, HEIGHT),
                                                      class_mode="binary",
                                                      batch_size=150)
```

```
Found 20000 validated image filenames belonging to 2 classes.
```

In [24]:
```
test_datagen = ImageDataGenerator(rescale=1./255)
test_datagenerator=test_datagen.flow_from_dataframe(dataframe=dataset_test,
                                                    x_col="image_path",
                                                    y_col="target",
                                                    target_size=(WIDTH, HEIGHT),
                                                    class_mode="binary",
                                                    batch_size=150)
```

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

Making CNN Model



Yan LeCun and Yoshua Bengoi introduced a Convolutional Neural Network in the Year 1995. Later on, they proved a remarkable achievement because they showed exceptional results in the domain of Images

A convolutional neural network has three parts. .

1 Convolutional Layer

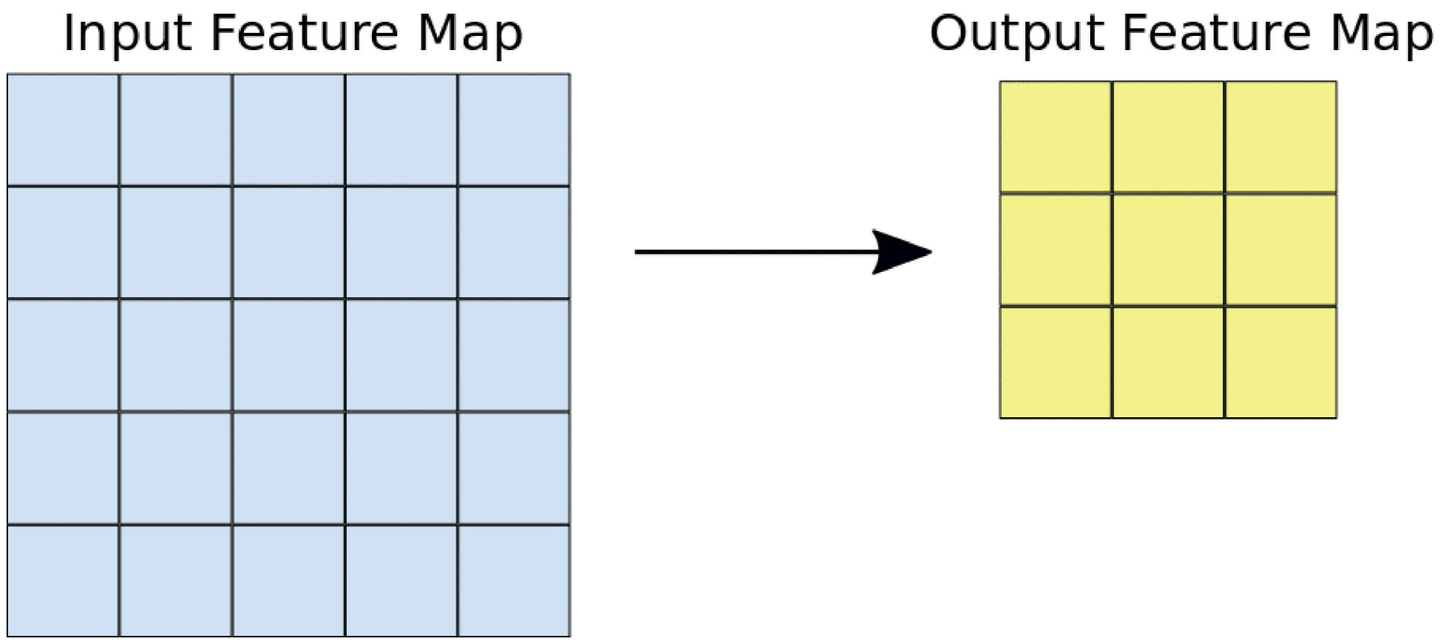
2 Pooling Layer

3 Fully Connected Layer

Convolutional layer

A convolutional layer help to extract the information from the image with the help of filter.

Please have a look the following image.



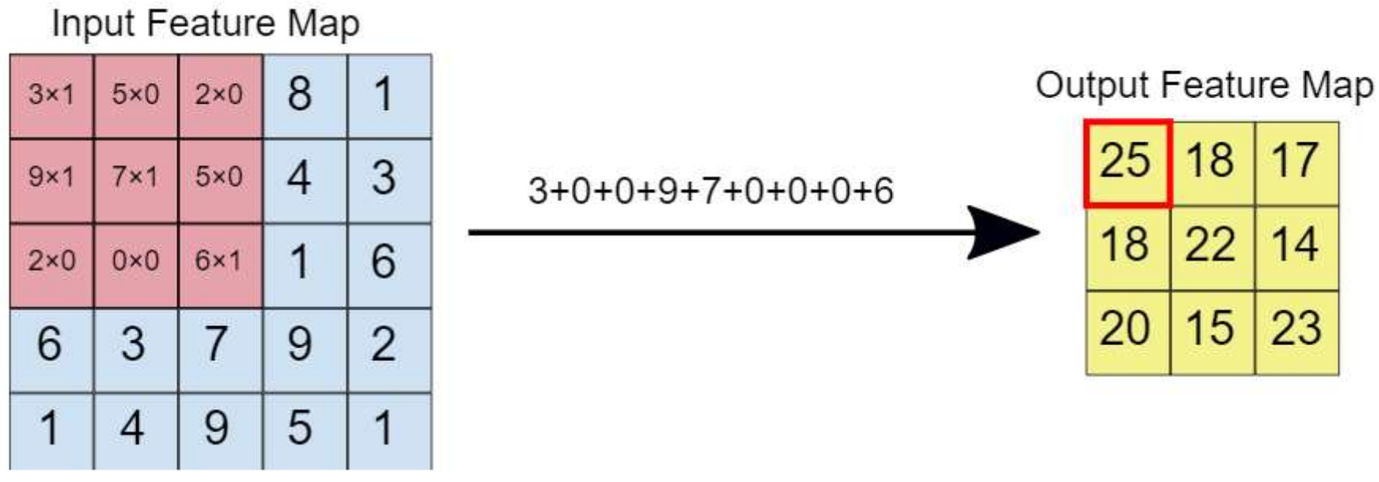
Feature Map

It will Take filter and map into image into size of 3 *3.

Then It will do some mathematical operation .

The Elements get Multiplied and then added to get one output and that out put will save in output feature map .

Below Image will clearfiy alot of thing

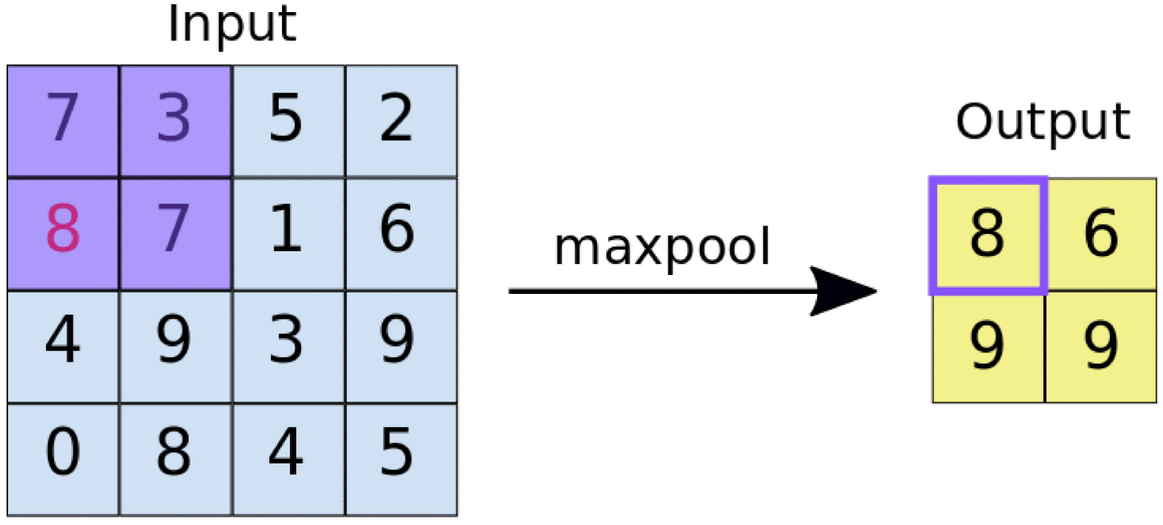


Pooling layer

The Intitution behind the pooling layer is to reduce the dimension of feature map.

the 2*2 pooling layer will apply to the input image and take the maximum value at each slide

Following Image will clear the concept more easily



Fully Connected Layer

The final layer is fully connected layer

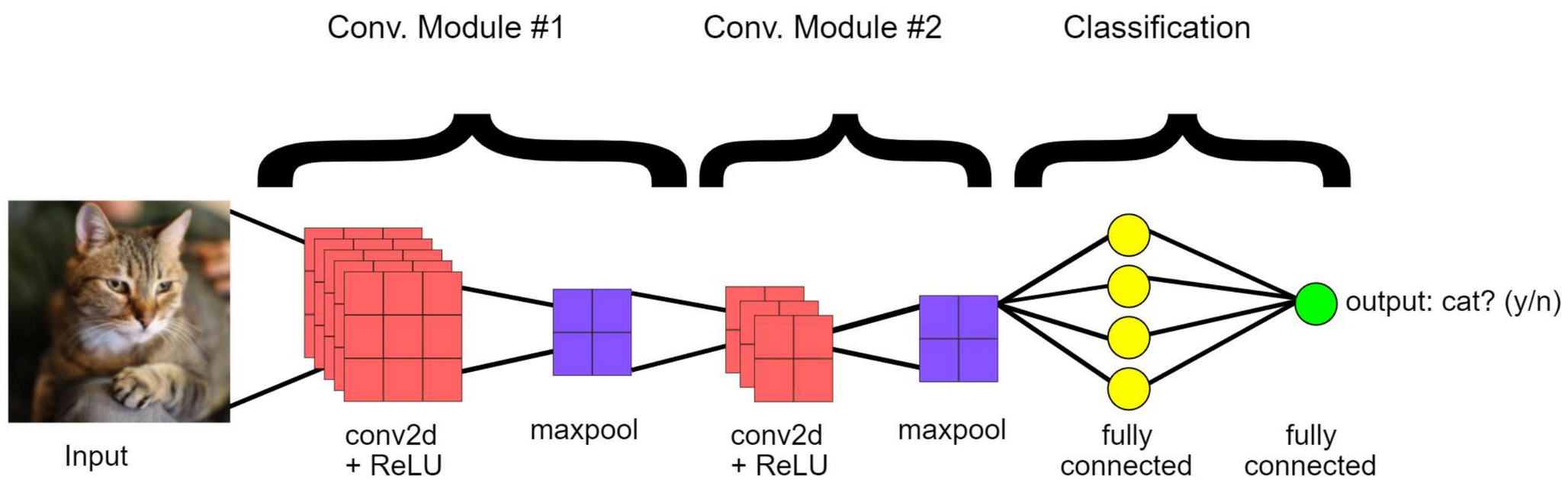
The input to the fully connected layer is the rich features that have been extracted using convolutional filters.

This will follow to the output layer where the probablities will be calculated of the image

Compile all of three

In [25]:
```
model = Sequential() # implement model layer
model.add(Conv2D(32, kernel_size=(3,3), input_shape=(WIDTH, HEIGHT, 3), activation='relu'))
model.add(Conv2D(64, kernel_size=(3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

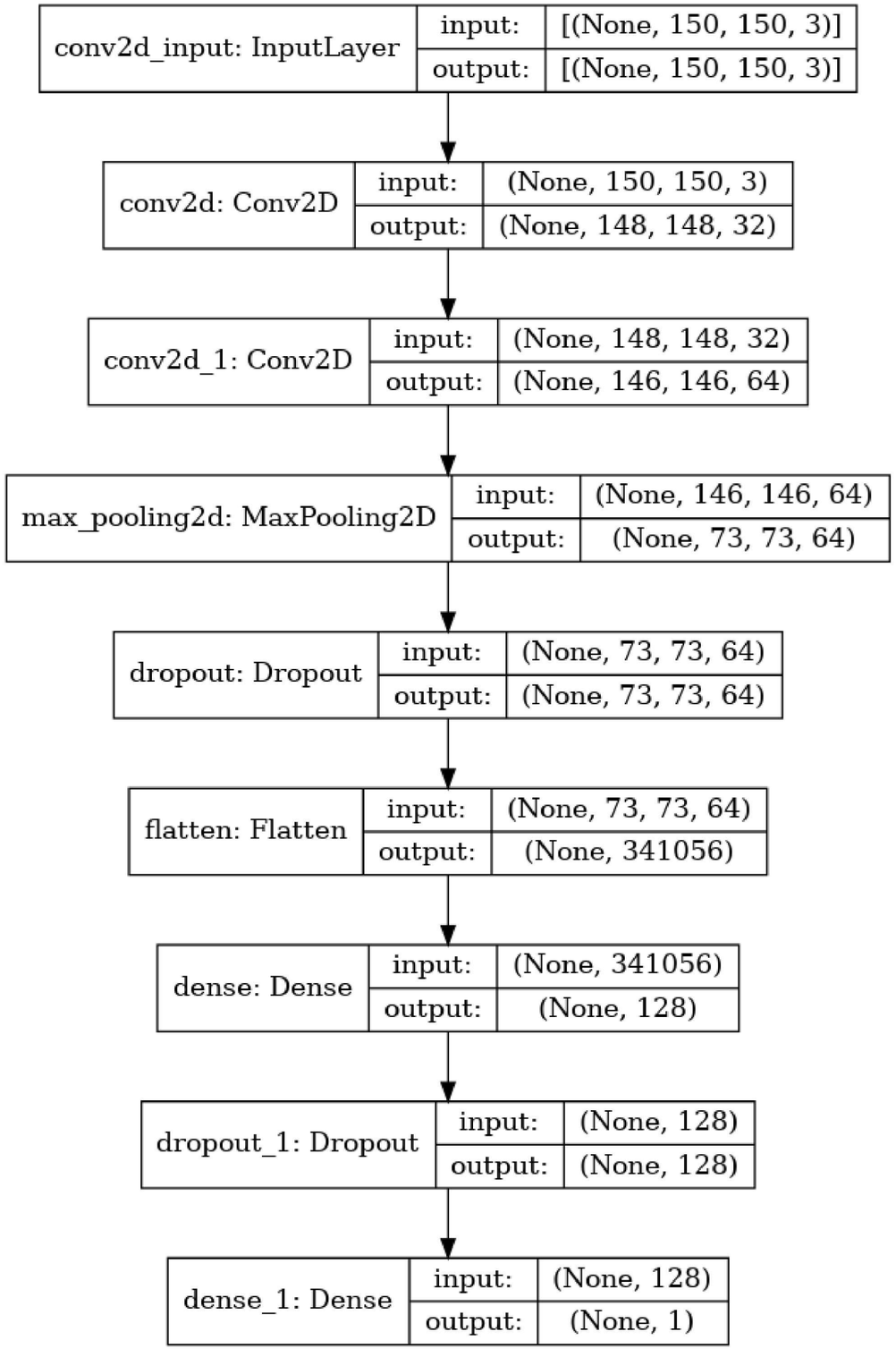
The Final Architecture looks like



Displaying The Model

In [26]:
```
from tensorflow.keras.utils import plot_model
from IPython.display import Image
plot_model(model, to_file='convnet.png', show_shapes=True,show_layer_names=True)
Image(filename='convnet.png')
```

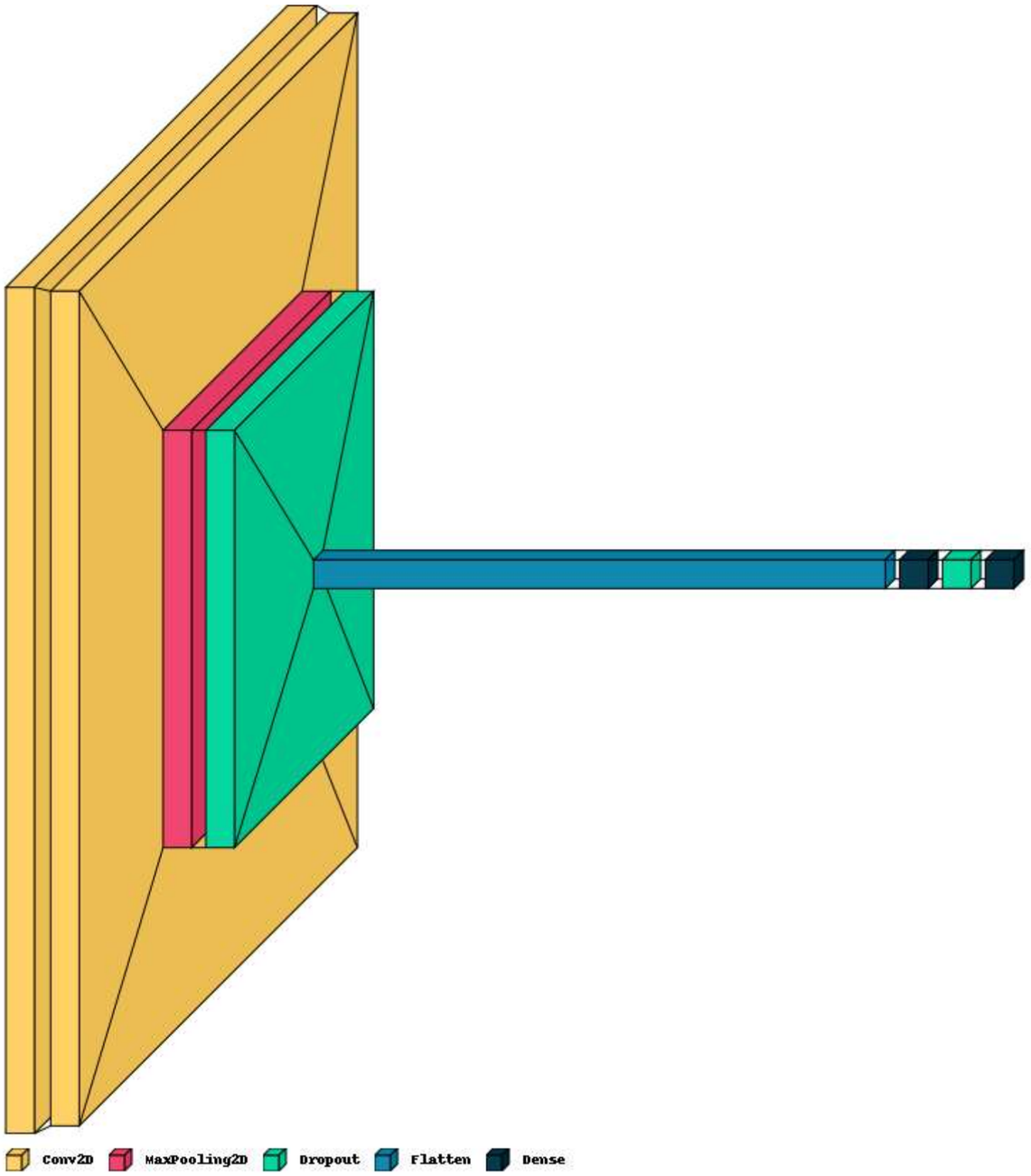
Out[26]:


conv2d_input: InputLayer
input: [(None, 150, 150, 3)]
output: [(None, 150, 150, 3)]
conv2d: Conv2D
input: (None, 150, 150, 3)
output: (None, 148, 148, 32)
conv2d_1: Conv2D
input: (None, 148, 148, 32)
output: (None, 146, 146, 64)
max_pooling2d: MaxPooling2D
input: (None, 146, 146, 64)
output: (None, 73, 73, 64)
dropout: Dropout
input: (None, 73, 73, 64)
output: (None, 73, 73, 64)
flatten: Flatten
input: (None, 73, 73, 64)
output: (None, 341056)
dense: Dense
input: (None, 341056)
output: (None, 128)
dropout_1: Dropout
input: (None, 128)
output: (None, 128)
dense_1: Dense
input: (None, 128)
output: (None, 1)

In [27]:
```
!pip install visualkeras
import visualkeras
visualkeras.layered_view(model, legend=True)
```

```
Collecting visualkeras
  Downloading visualkeras-0.0.2-py3-none-any.whl (12 kB)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.7/site-packages (from visualkeras) (9.1.0)
Requirement already satisfied: numpy>=1.18.1 in /opt/conda/lib/python3.7/site-packages (from visualkeras) (1.21.6)
Collecting aggdraw>=1.3.11
  Downloading aggdraw-1.3.16-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (989 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 989.8/989.8 kB 26.9 MB/s eta 0:00:0000:01
Installing collected packages: aggdraw, visualkeras
Successfully installed aggdraw-1.3.16 visualkeras-0.0.2
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

Out[27]:



In [28]:
```
model.compile(loss="binary_crossentropy", optimizer='adam', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 146, 146, 64)      18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 73, 73, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 73, 73, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 341056)            0
_________________________________________________________________
dense (Dense)                (None, 128)               43655296
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 43,674,817
Trainable params: 43,674,817
Non-trainable params: 0
_________________________________________________________________
```

Train The Model

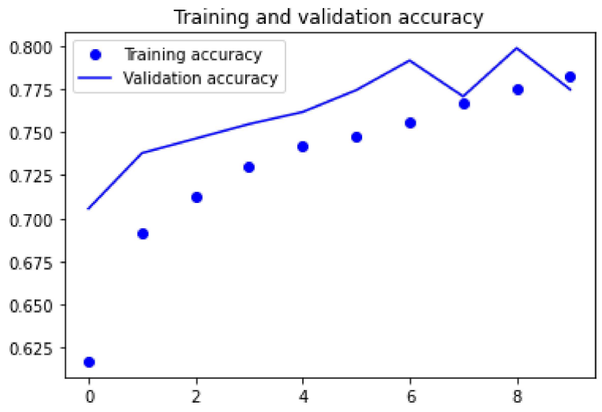
In [29]:
```
History=model.fit(train_datagenerator,
                  epochs=10,
                  validation_data=test_datagenerator,
                  validation_steps=dataset_test.shape[0]/150,
                  steps_per_epoch=dataset_train.shape[0]/150)
```

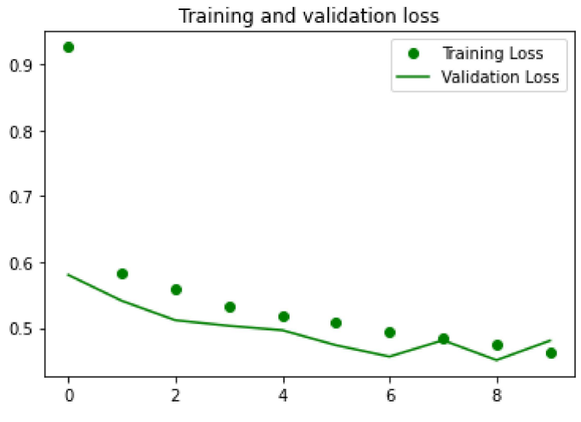
```
Epoch 1/10
133/133 [==============================] - 166s 1s/step - loss: 0.9261 - accuracy: 0.6172 - val_loss: 0.5804 - val_accuracy: 0.7056
Epoch 2/10
133/133 [==============================] - 157s 1s/step - loss: 0.5822 - accuracy: 0.6913 - val_loss: 0.5411 - val_accuracy: 0.7378
Epoch 3/10
133/133 [==============================] - 158s 1s/step - loss: 0.5582 - accuracy: 0.7122 - val_loss: 0.5118 - val_accuracy: 0.7462
Epoch 4/10
133/133 [==============================] - 157s 1s/step - loss: 0.5333 - accuracy: 0.7298 - val_loss: 0.5034 - val_accuracy: 0.7546
Epoch 5/10
133/133 [==============================] - 158s 1s/step - loss: 0.5183 - accuracy: 0.7415 - val_loss: 0.4966 - val_accuracy: 0.7616
Epoch 6/10
133/133 [==============================] - 157s 1s/step - loss: 0.5096 - accuracy: 0.7478 - val_loss: 0.4739 - val_accuracy: 0.7742
Epoch 7/10
133/133 [==============================] - 157s 1s/step - loss: 0.4949 - accuracy: 0.7559 - val_loss: 0.4566 - val_accuracy: 0.7914
Epoch 8/10
133/133 [==============================] - 157s 1s/step - loss: 0.4842 - accuracy: 0.7666 - val_loss: 0.4814 - val_accuracy: 0.7706
Epoch 9/10
133/133 [==============================] - 157s 1s/step - loss: 0.4739 - accuracy: 0.7746 - val_loss: 0.4512 - val_accuracy: 0.7986
Epoch 10/10
133/133 [==============================] - 157s 1s/step - loss: 0.4623 - accuracy: 0.7824 - val_loss: 0.4808 - val_accuracy: 0.7746
```

This Notebook is for learning purposes and I am deliberately setting the epoch value to a minimum so you can see the concept .

Plotting Loss and Accuracy

In [30]:
```
acc = History.history['accuracy']
val_acc = History.history['val_accuracy']
loss = History.history['loss']
val_loss = History.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'go', label='Training Loss')
plt.plot(epochs, val_loss, 'g', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





Model Evaluation

In [31]:
```
test_loss, test_acc = model.evaluate(test_datagenerator, steps=len(test_datagenerator), verbose=1)
print('Loss: %.3f' % (test_loss * 100.0))
print('Accuracy: %.3f' % (test_acc * 100.0))
```

```
34/34 [==============================] - 15s 426ms/step - loss: 0.4808 - accuracy: 0.7746
Loss: 48.085
Accuracy: 77.460
```

Confusion Matrix

In [32]:
```
from sklearn.metrics import confusion_matrix
import itertools
```

In [33]:
```
predictions = model.predict(x=test_datagenerator, steps= len(test_datagenerator), verbose=0)
```

In [34]:
```
test_datagenerator.classes
```

```
Out[34]: [0,
 1,
 0,
 1,
 0,
 1,
 1,
 0,
 1,
 1,
 1,
 1,
 1,
 0,
 0,
 1,
 0,
 1,
 1,
 1,
 0,
 1,
 0,
 0,
 0,
 1,
 0,
 1,
 0,
 1,
 0,
 0,
 0,
 0,
 1,
 1,
 1,
 0,
 1,
 0,
 0,
 1,
 0,
 0,
 0,
 1,
 0,
 1,
 0,
 0,
 0,
 0,
 1,
 1,
 1,
 0,
 0,
 0,
 0,
 0,
```

1,
1,
1,
0,
1,
0,
1,
1,
1,
0,
1,
0,
0,
0,
1,
1,
1,
0,
1,
0,
1,
0,
1,
0,
0,
1,
0,
0,
1,
0,
0,
1,
0,
1,
1,
1,
1,
0,
0,
1,
1,
1,
1,

1,
1,
0,
0,
1,
0,
0,
1,
1,
1,
1,
0,
0,
0,
0,
0,
0,
1,
0,
1,
1,
1,
1,
0,
0,
0,
1,
1,
1,
0,
0,
0,
1,
0,
1,
0,
1,
1,
0,
0,
0,
1,
0,
0,
0,
1,
0,

0,
0,
0,
0,
0,
0,
1,
0,
1,
0,
0,
1,
1,
1,
1,
0,
0,
1,
1,
0,
0,
1,
1,
1,
0,
1,
1,
1,
1,
0,
0,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
0,
1,
0,
1,
0,
1,
1,
0,
1,
1,
0,
0,
0,
0,

0,
0,
0,
1,
1,
0,
1,
1,
0,
0,
0,
0,
1,
0,
1,
0,
0,
0,
1,
0,
0,
0,
1,
1,
1,
0,
0,
1,
1,
0,
1,
0,
0,
0,
1,
1,
1,
1,
0,
0,
0,
1,
1,
0,
1,
1,
0,
1,

0,
0,
0,
1,
1,
0,
1,
0,
1,
1,
1,
1,
1,
0,
0,
1,
1,
1,
0,
1,
0,
1,
1,
0,
0,
1,
1,
0,
1,
1,
1,
0,
1,
0,
1,
0,
0,
1,
1,
0,
0,
0,
1,
1,
0,
0,
0,
1,
0,
1,
1,
0,
1,
0,

```
1,
0,
0,
0,
0,
0,
1,
0,
1,
0,
1,
0,
0,
0,
0,
1,
0,
0,
1,
1,
1,
1,
1,
1,
1,
0,
1,
0,
0,
1,
0,
1,
1,
1,
1,
0,
1,
0,
0,
0,
```

0,
0,
0,
0,
0,
1,
1,
1,
1,
0,
1,
0,
0,
1,
0,
1,
0,
0,
0,
1,
0,
0,
0,
1,
1,
1,
1,
1,
1,
0,
0,
0,
0,
0,
1,
1,
1,
0,
0,
1,
0,
0,
1,
0,
0,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
0,
0,
0,
0,
0,
1,
1,
1,
1,
0,
1,
0,
0,

0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
0,
1,
0,
0,
0,
1,
1,
0,
1,
1,
0,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
0,
1,
1,
0,
1,
1,
0,
0,
1,
1,
0,

0,
1,
1,
1,
1,
1,
1,
1,
0,
0,
1,
0,
1,
0,
1,
1,
1,
0,
1,
0,
0,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
0,
0,
1,
1,
1,
1,
1,
0,
0,
0,
1,
0,
0,
1,
1,
0,
1,
0,
1,
1,
1,
1,
0,
0,
0,
0,
1,
0,
0,

0,
0,
1,
1,
1,
0,
1,
0,
0,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
1,
1,
1,
0,
1,
0,
1,
1,
1,
1,
1,
1,
0,
1,
1,
0,
0,
0,
0,
0,
1,
0,
0,
1,

0,
0,
1,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
0,
0,
1,
0,
0,
1,
1,
1,
0,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
0,
0,
0,
0,
1,
1,
1,
0,
0,
0,
0,
0,
1,
1,
0,
0,
1,
1,
1,
1,
1,
0,
1,
0,
0,
0,
1,

1,
0,
1,
1,
0,
1,
0,
0,
0,
0,
1,
1,
0,
0,
0,
1,
0,
0,
0,
0,
0,
1,
1,
0,
1,
0,
1,
1,
1,
0,
1,
1,
0,
1,
0,
1,
1,
0,
0,
0,
1,
1,
1,
1,
0,

0,
0,
0,
0,
0,
0,
1,
0,
1,
1,
0,
0,
0,
1,
1,
0,
1,
1,
1,
1,
0,
1,
1,
1,
0,
1,
0,
0,
0,
1,
0,
1,
0,
1,
0,
1,
1,
0,
0,
0,
0,
0,
0,
0,
1,
0,
1,
1,
0,
0,
0,
0,
0,
0,
0,
1,
1,
0,
0,
1,
0,
1,
1,

1,
0,
0,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
0,
0,
1,
1,
0,
0,
0,
0,
1,
1,
1,
0,
0,
0,
1,
1,
0,
0,
0,
1,
0,
0,
0,
1,
0,
1,
1,
0,
1,
1,
0,
0,
1,
1,
0,
0,
1,
0,
1,
1,
1,
0,
0,
1,
0,
1,
1,
0,

0,
0,
1,
1,
1,
0,
1,
1,
0,
1,
0,
1,
0,
0,
0,
1,
1,
1,
1,
0,
0,
1,
1,
0,
0,
0,
1,
0,
1,
0,
0,
1,
1,
1,
0,
0,
1,
0,
1,
0,
0,

```
0,
0,
0,
0,
0,
0,
1,
0,
0,
1,
1,
0,
1,
1,
0,
1,
1,
0,
0,
1,
1,
1,
1,
1,
0,
1,
0,
0,
1,
1,
0,
0,
1,
1,
0,
0,
0,
...]
```

In [35]:
```
cm = confusion_matrix(y_true=test_datagenerator.classes, y_pred=np.argmax(predictions, axis=-1))
```

In [36]:
```
def plot_confusion_matrix(cm, classes,normalize=False,title='Confusion matrix',cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
        print(cm)
```

```
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
        horizontalalignment="center",
        color="white" if cm[i, j] > thresh else "black")
        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
```

In [37]:
```
cm_plot_labels = ['0_cat', '1_dog']

plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion Matrix')
```

```
Confusion matrix, without normalization
[[2515    0]
 [2485    0]]
```