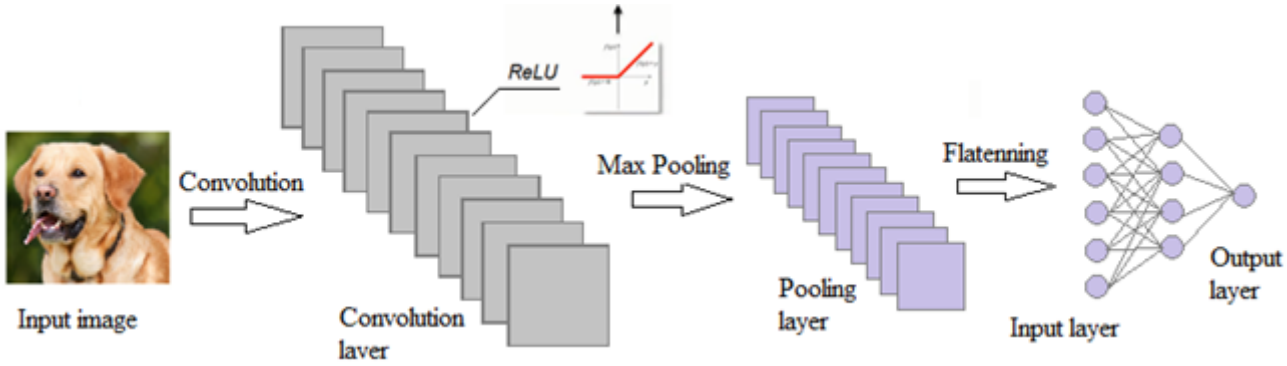


Pet Classification Project with CNN

In [1]:
```
# Load necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Flatten, Conv2D, Dropout, MaxPooling2D
from keras.models import Sequential
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model
```

Step 1 : Prepare Data

Cats Train images folder :- D:\AI Engineer Masters\3.Project 3 - Pet Classification\train\cats
Dogs Train images folder :- D:\AI Engineer Masters\3.Project 3 - Pet Classification\train\dogs
Cats Test images folder :- D:\AI Engineer Masters\3.Project 3 - Pet Classification\test\cats
Dogs Test images folder :- D:\AI Engineer Masters\3.Project 3 - Pet Classification\test\dogs

Here we have less images for the training set, which is not enough to avoid over-fitting. So, we perform image augmentation, such as rotating, flipping, or shearing to increase the number of images. It splits training images into batches, and each batch will be applied random image transformation on a random selection of images, to create many more diverse images.

Specifically, we will use flow_from_directory(directory) method from Keras Official website to load images and apply augmentation. This is why we structured the data folders in a specific way so that the class of each image can be identified from its folder name. The below code snippet allows us to augment images and fit and test CNN.

In [2]:
```
# create train data generator for scaling. For training , setting the pictures in proper similar size & flip
# The problem is that images may have different formats and image size. So, we need to convert images into the same format and
 fixed size.
# We can also do width_shift_range as well as height_shift_range
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)

# create train data generator for scaling
test_datagen=ImageDataGenerator(rescale=1./255)

# preparing data set with size 64 X 64 for ease of training
# 64X64 could be enough for good accuracy. Fwe can use higher dimensions like (128, 128) if we have more processing power.
train_set=train_datagen.flow_from_directory(r'D:\AI Engineer Masters\3.Project 3 - Pet Classification\train', target_size=(64,6
4),batch_size=64,class_mode='binary')
test_set=test_datagen.flow_from_directory(r'D:\AI Engineer Masters\3.Project 3 - Pet Classification\test', target_size=(64,64),
batch_size=64,class_mode='binary')
```

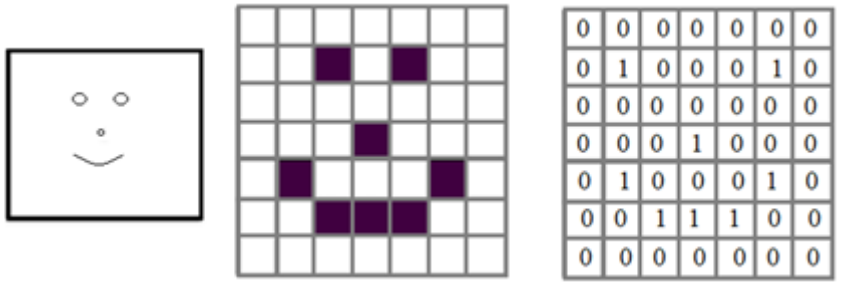
```
Found 20040 images belonging to 2 classes.
Found 5000 images belonging to 2 classes.
```

Step 2 : Model building

In general, 4 steps are required to build a CNN: Convolution, Max pooling, Flattening, and Full connection

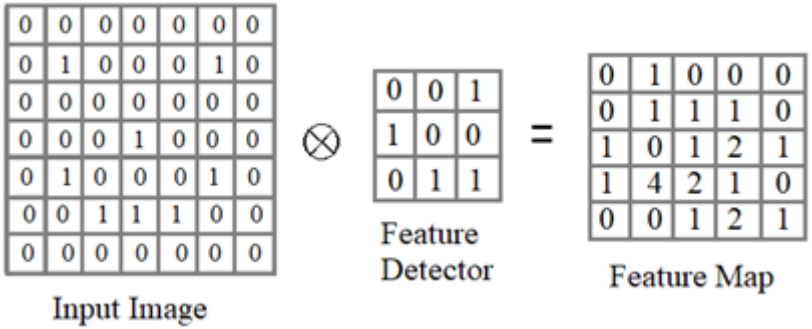
2.1 Convolution

Conceptually, convolution is to apply feature detectors on the input image. To simplify the concept, take a smiling face as an input image, which is represented as an array of 0 and 1.



0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

The feature detector is also an array of numbers. For each feature detector, we slide it over the image and produce a new array of numbers, representing a feature of the image. So, the operation between an input image and a feature detector that results in a feature map is Convolution as shown below



0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

Input Image

⊗

0	0	1
1	0	0
0	1	1

Feature Detector

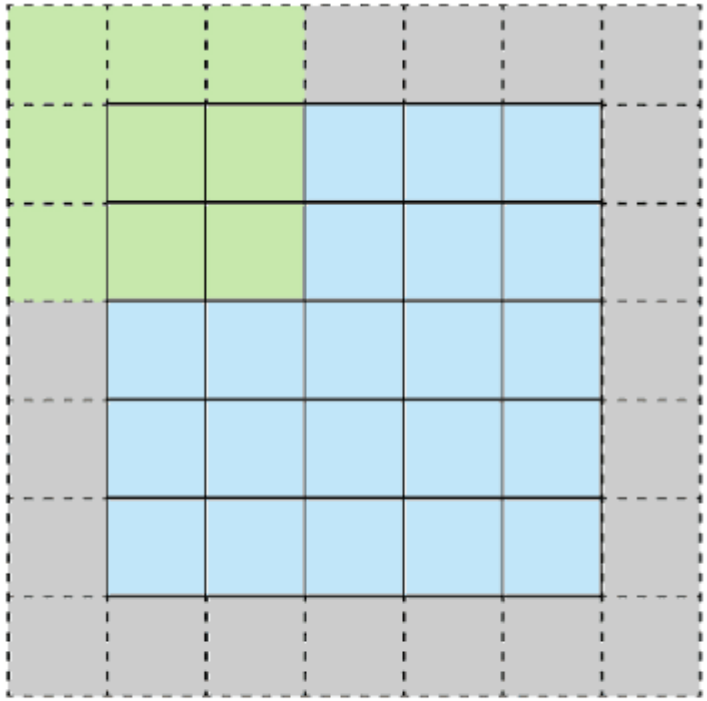
=

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

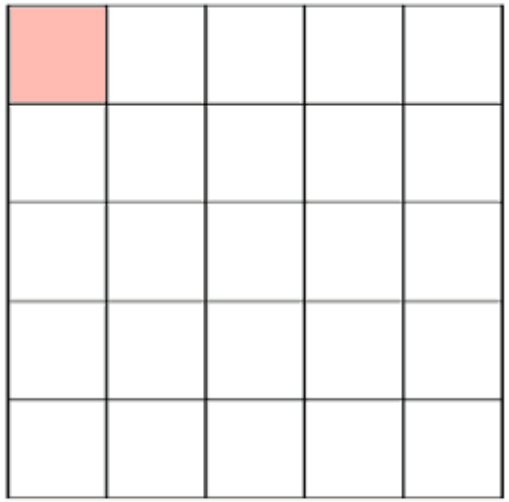
Feature Map

If repeating the above convolution with different feature detectors, we produce as many feature maps as feature detectors, obtaining a convolution layer.

Now, stride denotes how many steps we are moving in each steps in convolution.By default it is one.
To maintain the dimension of output as in input , we use padding. Padding is a process of adding zeros to the input matrix symmetrically. In the following example,the extra grey blocks denote the padding. It is used to make the dimension of output same as input.



Stride 1 with Padding

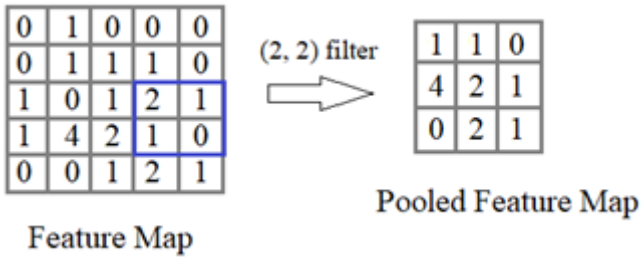


Feature Map

In [3]:
```
# We can use Conv2D() function from Keras to build the first convolution layer.
# Note:The number of feature detectors is set to be 32, and its dimension is (3, 3).
# In most CNN architectures, a common practice is to start with 32 feature detectors and increase to 64 or 128 if needed.
# input_shape is the shape of input images on which we apply feature detectors through convolution.
# We have set the imgae size to 64x64 and so we are puttign the input_shape as (64, 64, 3).
# Here, 3 is the number of channels for a colored image, (64, 64) is the image dimension for each channel.
# We can use padding as same to keep the same dimension of the input to avoid losing any data.
# Another argument is the activation function. we use ReLU to remove any negative pixel values in feature maps.
# This is because depending on the parameters used in convolution, we may obtain negative pixels in feature maps.
# Removing negative pixels adds non-linearity for a non-linear classification problem.
# The final argument is kernel_initializer. The main purpose of it is to initialize the weight matrix in the neural network.
# kernel_initializer='he_uniform' works well with relu activation
model=Sequential()
model.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(64,64,3),padding='Same',activation='relu',kernel_initializer='he_uni
form'))
```

2.2 Max pooling

Max pooling is to reduce the size of a feature map by sliding a table, for example (2,2), and taking the maximum value in the table. If we slide a table with a stride of 2 over 1 feature map of (5,5), we get a feature map with reduced size of (3,3).



Repeating max pooling on each feature map produces a pooling layer. Fundamentally, max pooling is to reduce the number of nodes in the fully connected layers without losing key features and spatial structure information in the images.

In [4]:
```
# We can use MaxPooling2D() function to add the pooling layer. In general, we use a 2x2 filter for pooling.
# We can also use a different strides if we want like strides=(2,2)
model.add(MaxPooling2D(pool_size=(2,2)))
```

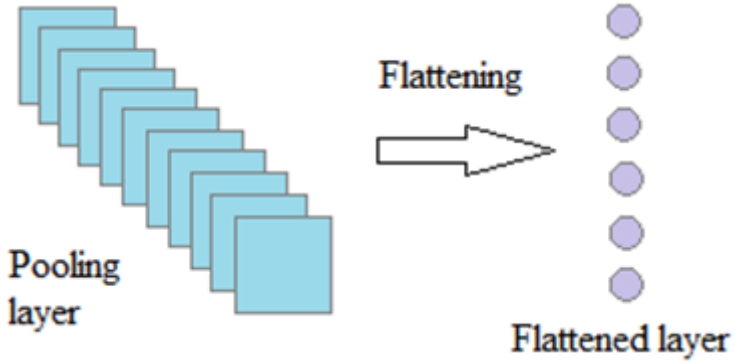
2.3 Dropout

Simply put, dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. By "ignoring", I mean these units are not considered during a particular forward or backward pass.
A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the

In [5]:
```
# We can use Dropout() function for dropout.
# The default interpretation of the dropout hyperparameter is the probability of training a given node in a layer, where 1.0 me
ans no dropout, and 0.0 means no outputs from the layer.
# A good value for dropout in a hidden layer is between 0.5 and 0.8. Input layers use a larger dropout rate, such as of 0.8.
model.add(Dropout(0.25))
```

2.4 Flattening

Flattening is to take all pooled feature maps into a single vector as the input for the fully connected layers as shown below



With convolution, we get many feature maps, each of which represents a specific feature of the image. Thus, each node in the flattened vector will represent a specific detail of the input image.

In [6]:
```
# We can use Flatten() function to do the same
model.add(Flatten())
```

2.5 Full connection

With the above, we converted an image into a one-dimensional vector. Now we will build a classifier using this vector as the input layer. First, create a hidden layer. output_dim is the number of nodes in the hidden layer.

In [7]:
```
# We can use Dense() function to add a hidden layer
# As a common practice, we choose 128 to start with and use ReLU as the activation function.
model.add(Dense(units=128,activation='relu',kernel_initializer='he_uniform'))
# Then lets add an output layer. For binary classification, units is 1, and the activation function is Sigmoid.
# And GlorotNormal kernal initializer works well with sigmoid activation
model.add(Dense(units=1,activation='sigmoid',kernel_initializer='GlorotNormal'))
```

Step 3 : Model Compiling

With all layers added, let's compile the CNN by choosing an SGD algorithm, a loss function, and performance metrics.

In [8]:
```
# We use binary_crossentropy for binary classification, and use categorical_crossentropy for multiple classification problem as
loss function.
# One interesting and dominant argument about optimizers is that SGD better generalizes than Adam.
# These papers argue that although Adam converges faster, SGD generalizes better than Adam and thus results in improved final p
erformance.
# However, I m still using adam as it's much faster than SGD for the time being.
model.compile(optimizer='adam',loss='binary_crossentropy',metrics='accuracy')
```

Step 4 : Model Fitting

In [9]:
```
# We can use fit method to do it.
# steps_per_epoch: Total number of steps (batches of samples) to yield from generator before declaring one epoch finished and s
tarting the next epoch.
# It should typically be equal to the number of unique samples of your dataset divided by the batch size.
# In our case we have 20040 train samples and batch size is 64
# validation_steps are similar to steps_per_epoch but it is on the validation data instead of the training data.
# In our case we have 5000 test samples and batch size is 64
hist=model.fit(train_set,steps_per_epoch=20040/64,epochs=10,validation_data=test_set,validation_steps=5000/64,verbose=0)
```

Step 5 : Model Evaluation

In [10]:
```
# evaluate model
_,acc=model.evaluate(train_set,steps=20040/64,verbose=0)
print('Train Accuracy > %.3f' % (acc * 100.0))
_,acc=model.evaluate(test_set,steps=5000/64,verbose=0)
print('Test Accuracy > %.3f' % (acc * 100.0))
```

```
Train Accuracy > 80.015
Test Accuracy > 77.340
```

Step 6 : Improve the Model

1 Block VGG Model

In [11]:
```
# The current model we build is basically a VGG 1 Block Model
model.summary()
print('1 Block VGG Model ::: Accuracy > %.3f' % (acc * 100.0))
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 128)               4194432
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 4,195,457
Trainable params: 4,195,457
Non-trainable params: 0
_________________________________________________________________
1 Block VGG Model ::: Accuracy > 77.340
```

2 Block VGG Model

Adding a 64 filter convoluation layer

In [12]:

```
model=Sequential()
model.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(64,64,3),padding='Same',activation='relu',kernel_initializer='he_uni
form'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding='Same',activation='relu',kernel_initializer='he_uniform'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(units=128,activation= 'relu',kernel_initializer='he_uniform'))
model.add(Dense(units=1,activation='sigmoid',kernel_initializer='GlorotNormal'))
model.compile(optimizer='adam',loss='binary_crossentropy',metrics='accuracy')
model.summary()
# fit model
model.fit(train_set,steps_per_epoch=20040/64,epochs=10,validation_data=test_set,validation_steps=5000/64,verbose=0)
# evaluate model
_,acc=model.evaluate(train_set,steps=20040/64,verbose=0)
print('Train Accuracy > %.3f' % (acc * 100.0))
_,acc=model.evaluate(test_set, steps=5000/64,verbose=0)
print('Test Accuracy > %.3f' % (acc * 100.0))
print('2 Block VGG Model ::: Accuracy > %.3f' % (acc * 100.0))
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 64, 64, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               2097280
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 129
=================================================================
Total params: 2,116,801
Trainable params: 2,116,801
Non-trainable params: 0
_________________________________________________________________
Train Accuracy > 50.000
Test Accuracy > 50.000
2 Block VGG Model ::: Accuracy > 50.000
```

3 Block VGG Model

Adding a 128 filter convoluation layer

In [13]:

```
model=Sequential()
model.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(64,64,3),padding='Same',activation='relu',kernel_initializer='he_uni
form'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding='Same',activation='relu',kernel_initializer='he_uniform'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Conv2D(filters=128,kernel_size=(3,3),padding='Same',activation='relu',kernel_initializer='he_uniform'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(units=128,activation= 'relu',kernel_initializer='he_uniform'))
model.add(Dense(units=1,activation='sigmoid',kernel_initializer='GlorotNormal'))
model.compile(optimizer='adam',loss='binary_crossentropy',metrics='accuracy')
model.summary()
# fit model
model.fit(train_set,steps_per_epoch=20040/64,epochs=10,validation_data=test_set,validation_steps=5000/64,verbose=0)
# evaluate model
_,acc=model.evaluate(train_set,steps=20040/64,verbose=0)
print('Train Accuracy > %.3f' % (acc * 100.0))
_,acc=model.evaluate(test_set, steps=5000/64,verbose=0)
print('Test Accuracy > %.3f' % (acc * 100.0))
print('3 Block VGG Model ::: Accuracy > %.3f' % (acc * 100.0))
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 64, 64, 32)        896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               1048704
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,142,081
Trainable params: 1,142,081
Non-trainable params: 0
_________________________________________________________________
Train Accuracy > 82.176
Test Accuracy > 81.120
3 Block VGG Model ::: Accuracy > 81.120
```

Transfer Learning

Transfer learning involves using all or parts of a model trained on a related task.
Keras provides a range of pre-trained models that can be loaded and used wholly or partially via the Keras Applications API.

Using VGG-16 Model

A useful model for transfer learning is one of the VGG models, such as VGG-16 with 16 layers that at the time it was developed, achieved top results on the ImageNet photo classification challenge.
The model is comprised of two main parts, the feature extractor part of the model that is made up of VGG blocks, and the classifier part of the model that is made up of fully connected layers and the output layer.
We can use the feature extraction part of the model and add a new classifier part of the model that is tailored to the dogs and cats dataset. Specifically, we can hold the weights of all of the convolutional layers fixed during training, and only train new fully connected layers that will learn to interpret the features extracted from the model and make a binary classification.

In [14]:
```
# This can be achieved by loading the VGG-16 model, removing the fully connected layers from the output-end of the model,
# then adding the new fully connected layers to interpret the model output and make a prediction.
# The classifier part of the model can be removed automatically by setting the “include_top” argument to “False“,
# which also requires that the shape of the input also be specified for the model, in this case (64, 64, 3).
# This means that the loaded model ends at the last max pooling layer, after which we can manually add a Flatten layer and the
 new clasifier layers.

# The VGG16 model was trained on a specific ImageNet challenge dataset.
# As such, it is configured to expected input images to have the shape 224×224 pixels.
# However, considering the processing power , I am still keeping it 64X64 as it is for learning purpose.

# Load model
model=VGG16(include_top=False,input_shape=(64,64,3))
# Mark loaded layers as not trainable
for layer in model.layers:
    layer.trainable = False
# Add new classifier layers
flat1=Flatten()(model.layers[-1].output)
class1=Dense(units=128,activation='relu',kernel_initializer='he_uniform')(flat1)
output=Dense(units=1,activation='sigmoid',kernel_initializer='GlorotNormal')(class1)
# Define new model
model=Model(inputs=model.inputs,outputs=output)
# Compile model
model.compile(optimizer='adam',loss='binary_crossentropy',metrics='accuracy')
model.summary()
# fit model
model.fit(train_set,steps_per_epoch=20040/64,epochs=10,validation_data=test_set,validation_steps=5000/64,verbose=0)
# evaluate model
_,acc=model.evaluate(train_set,steps=20040/64,verbose=0)
print('Train Accuracy > %.3f' % (acc * 100.0))
_,acc=model.evaluate(test_set, steps=5000/64,verbose=0)
print('Test Accuracy > %.3f' % (acc * 100.0))
print('Transfer Learning VGG-16 Model ::: Accuracy > %.3f' % (acc * 100.0))
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 64, 64, 3)]	0
block1_conv1 (Conv2D)	(None, 64, 64, 64)	1792
block1_conv2 (Conv2D)	(None, 64, 64, 64)	36928
block1_pool (MaxPooling2D)	(None, 32, 32, 64)	0
block2_conv1 (Conv2D)	(None, 32, 32, 128)	73856
block2_conv2 (Conv2D)	(None, 32, 32, 128)	147584
block2_pool (MaxPooling2D)	(None, 16, 16, 128)	0
block3_conv1 (Conv2D)	(None, 16, 16, 256)	295168
block3_conv2 (Conv2D)	(None, 16, 16, 256)	590080
block3_conv3 (Conv2D)	(None, 16, 16, 256)	590080
block3_pool (MaxPooling2D)	(None, 8, 8, 256)	0
block4_conv1 (Conv2D)	(None, 8, 8, 512)	1180160
block4_conv2 (Conv2D)	(None, 8, 8, 512)	2359808
block4_conv3 (Conv2D)	(None, 8, 8, 512)	2359808
block4_pool (MaxPooling2D)	(None, 4, 4, 512)	0
block5_conv1 (Conv2D)	(None, 4, 4, 512)	2359808
block5_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block5_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block5_pool (MaxPooling2D)	(None, 2, 2, 512)	0
flatten_3 (Flatten)	(None, 2048)	0

```
dense_6 (Dense)              (None, 128)               262272
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 129
=================================================================
Total params: 14,977,089
Trainable params: 262,401
Non-trainable params: 14,714,688
_________________________________________________________________
Train Accuracy > 84.701
Test Accuracy > 81.960
Transfer Learning VGG-16 Model ::: Accuracy > 81.960
```

01 Block VGG Model ::: Accuracy > 77.340
02 Block VGG Model ::: Accuracy > 50.000
03 Block VGG Model ::: Accuracy > 81.120
16 Block VGG Model ::: Accuracy > 81.960

We got some what a decent model w.r.t. the processing power limitations
So, lets go ahead with VGG-16 Model.

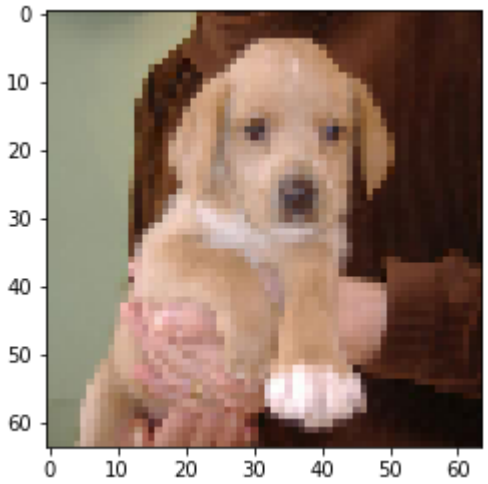
Step 7 : Save the Model

In [15]:
```
# Its important to save the model for future use.
model.save(r'D:\AI Engineer Masters\3.Project 3 - Pet Classification\VGG16_final_model.h5')
```

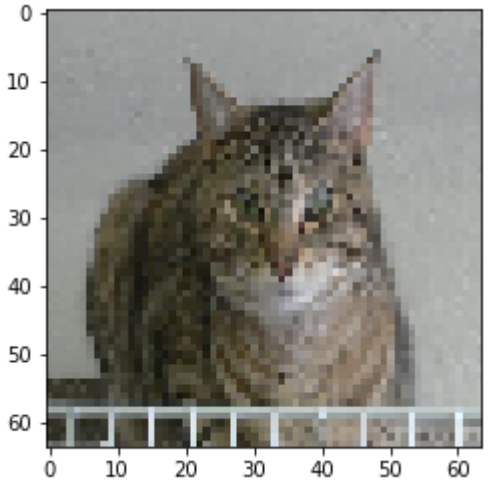
Step 8 : Make predictions

In [16]:

```
# Load model
model = load_model(r'D:\AI Engineer Masters\3.Project 3 - Pet Classification\VGG16_final_model.h5')

# load the image
fileName=r'D:\AI Engineer Masters\3.Project 3 - Pet Classification\Val\dogs\103.jpg'
img = load_img(fileName, target_size=(64, 64))
# Convert to array
img = img_to_array(img)
plt.imshow(img/255.)
plt.show()
# Reshape into a single sample with 3 channels
img = img.reshape(1, 64, 64, 3)

# Predict the class
result = model.predict(img)
if result[0] == 1 : print(fileName,'is a dog image')
if result[0] == 0 : print(fileName,'is a cat image')

# load the image
fileName=r'D:\AI Engineer Masters\3.Project 3 - Pet Classification\Val\cats\103.jpg'
img = load_img(fileName, target_size=(64, 64))
# Convert to array
img = img_to_array(img)
plt.imshow(img/255.)
plt.show()
# Reshape into a single sample with 3 channels
img = img.reshape(1, 64, 64, 3)

# Predict the class
result = model.predict(img)
if result[0] == 1 : print(fileName,'is a dog image')
if result[0] == 0 : print(fileName,'is a cat image')
```



D:\AI Engineer Masters\3.Project 3 - Pet Classification\Val\dogs\103.jpg is a dog image



D:\AI Engineer Masters\3.Project 3 - Pet Classification\Val\cats\103.jpg is a cat image