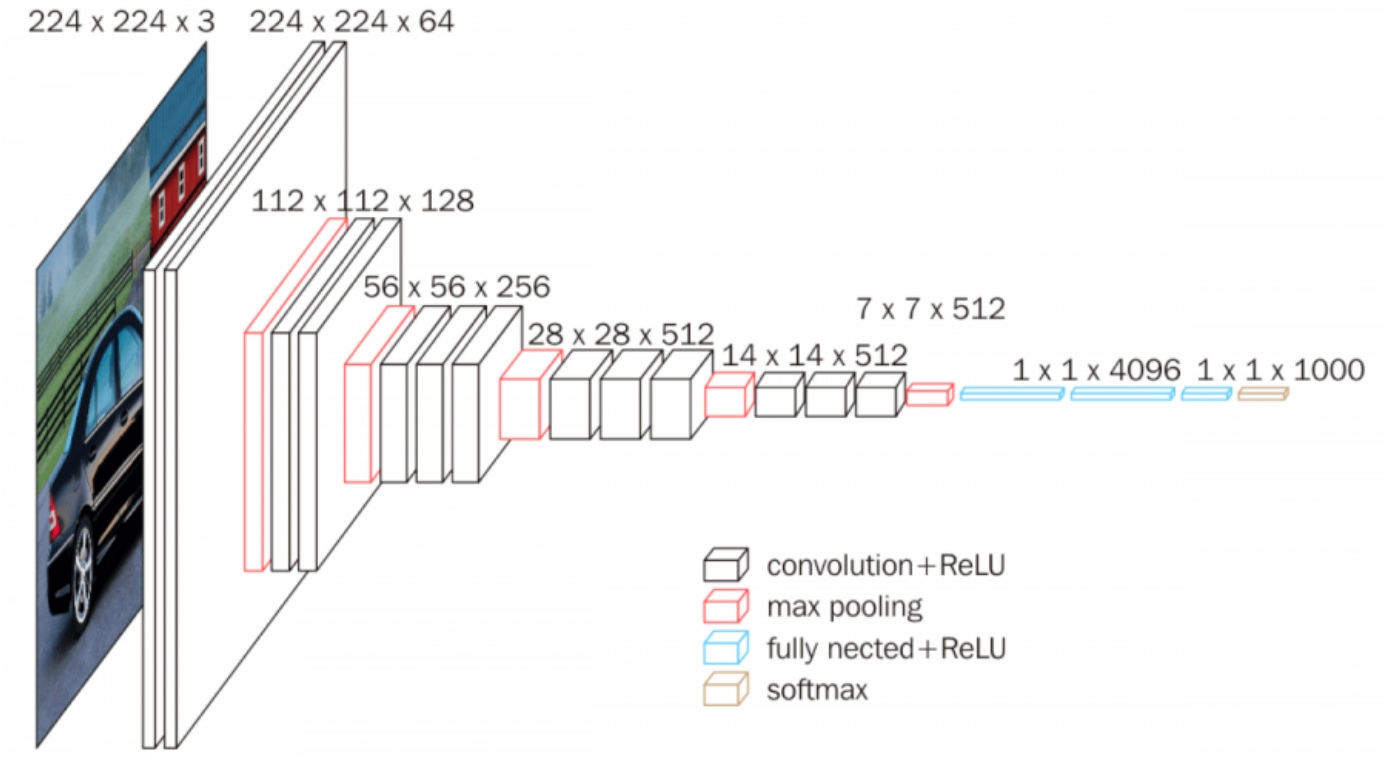
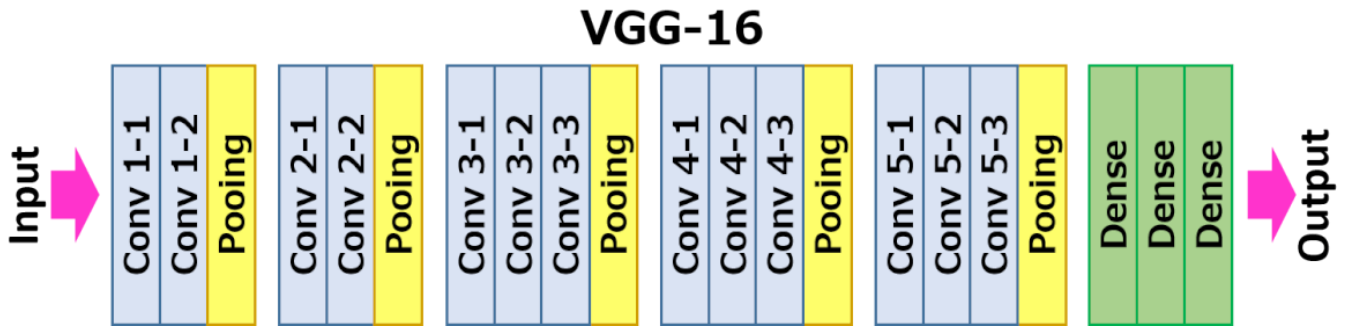
VGG16

- VGG16 is a convolution neural net (CNN) architecture which was used to win ILSVR(Imagenet) competition in 2014.
- It is considered to be one of the excellent vision model architecture till date. Most unique thing about VGG16 is that instead of having a large number of hyper-parameter they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2.
- It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a softmax for output. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network and it has about 138 million (approx) parameters.





The following are the layers of the model:

- Convolutional Layers = 13
- Pooling Layers = 5
- Dense Layers = 3

Let us explore the layers in detail:

1. **Input:** Image of dimensions (224, 224, 3).

2. **Convolution Layer Conv1:**

    ```
    - Conv1-1: 64 filters
    - Conv1-2: 64 filters and Max Pooling
    - Image dimensions: (224, 224)
    ```

3. **Convolution layer Conv2:** Now, we increase the filters to 128

    ```
    - Input Image dimensions: (112,112)
    - Conv2-1: 128 filters
    - Conv2-2: 128 filters and Max Pooling
    ```

4. **Convolution Layer Conv3:** Again, double the filters to 256, and now add another convolution layer

    ```
    - Input Image dimensions: (56,56)
    - Conv3-1: 256 filters
    - Conv3-2: 256 filters
    - Conv3-3: 256 filters and Max Pooling
    ```

5. **Convolution Layer Conv4:** Similar to Conv3, but now with 512 filters

    ```
    - Input Image dimensions: (28, 28)
    - Conv4-1: 512 filters
    - Conv4-2: 512 filters
    - Conv4-3: 512 filters and Max Pooling
    ```

6. **Convolution Layer Conv5:** Same as Conv4

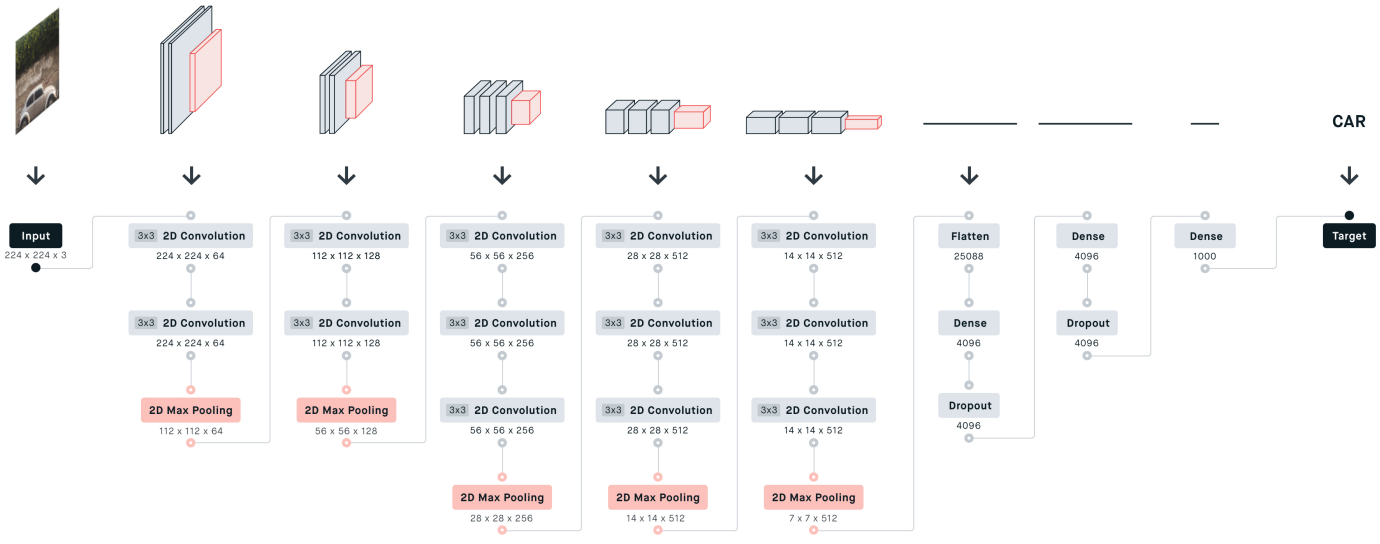
    ```
    - Input Image dimensions: (14, 14)
    - Conv5-1: 512 filters
    - Conv5-2: 512 filters
    - Conv5-3: 512 filters and Max Pooling
    - The output dimensions here are (7, 7). At this point, we flatten the output
    of this layer to generate a feature vector
    ```

7. **Fully Connected/Dense FC1:** 4096 nodes, generating a feature vector of size(1, 4096)
8. **Fully ConnectedDense FC2:** 4096 nodes generating a feature vector of size(1, 4096)
9. **Fully Connected /Dense FC3:** 4096 nodes, generating 1000 channels for 1000 classes. This is then passed on to a Softmax activation function
10. **Output layer**

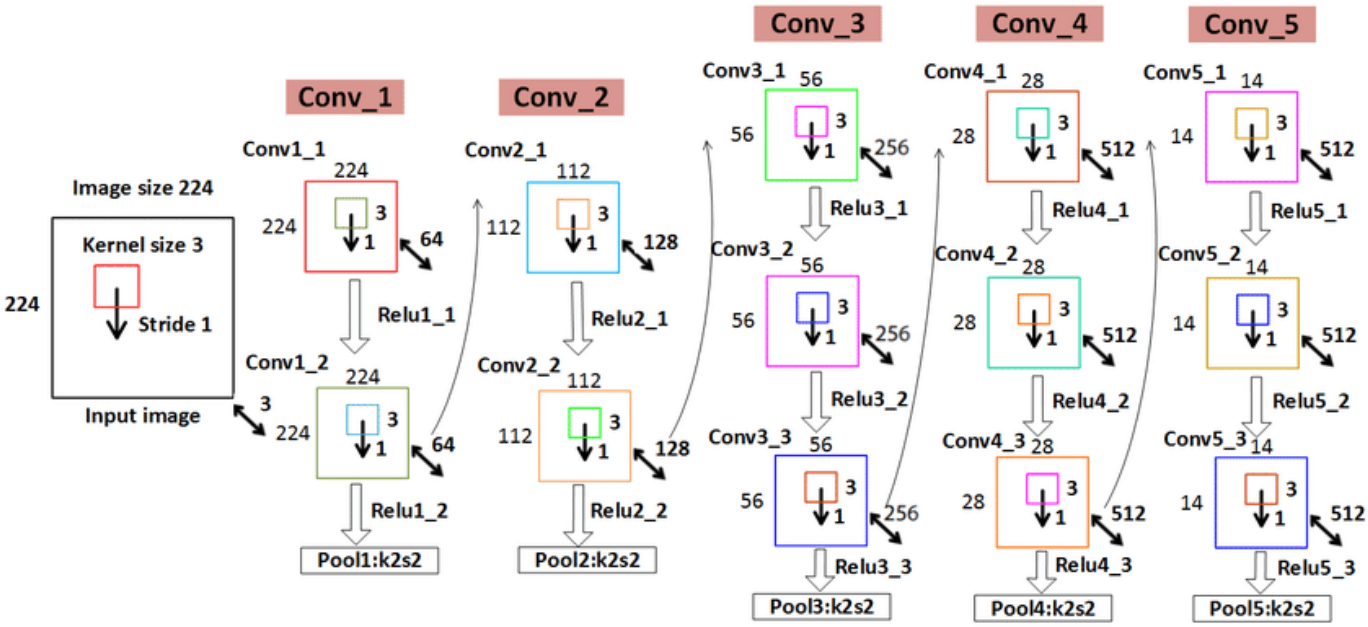
> VGG16 contains 16 layers and VGG19 contains 19 layers. A series of VGGs are exactly the same in the last three fully connected layers. The overall structure includes 5 sets of convolutional layers, followed by a MaxPool. The difference is that more and more cascaded convolutional layers are included in the five sets of convolutional layers .

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

Each convolutional layer in AlexNet contains only one convolution, and the size of the convolution kernel is 7 *7 ,. In VGGNet, each convolution layer contains 2 to 4 convolution operations. The size of the convolution kernel is 3* 3, the convolution step size is 1, the pooling kernel is 2 * 2, and the step size is 2. The most obvious improvement of VGGNet is to reduce the size of the convolution kernel and increase the number of convolution layers.



Using multiple convolution layers with smaller convolution kernels instead of a larger convolution layer with convolution kernels can reduce parameters on the one hand, and the author believes that it is equivalent to more non-linear mapping, which increases the Fit expression ability.



> Two consecutive 3 *3 convolutions are equivalent to a 5* 5 receptive field, and three are equivalent to 7 *7. The advantages of using three 3* 3 convolutions instead of one 7 *7 convolution are twofold : one, including three ReLu layers instead of one , makes the decision function more discriminative; and two, reducing parameters . For example, the input and output are all C channels. 3 convolutional layers using 3* 3 require 3 (3 *3* C *C) = 27* C *C, and 1 convolutional layer using 7* 7 requires 7 *7* C *C = 49C* C. This can be seen as applying a kind of regularization to the 7 *7 convolution, so that it is decomposed into three 3* 3 convolutions.
>
> The 1 *1 convolution layer is mainly to increase the non-linearity of the decision function without affecting the receptive field of the convolution layer. Although the 1* 1 convolution operation is linear, ReLu adds non-linearity.

Some basic questions

Q1: Why can 3 3x3 convolutions replace 7x7 convolutions?

Answer 1

3 3x3 convolutions, using 3 non-linear activation functions, increasing non-linear expression capabilities, making the segmentation plane more separable Reduce the number of parameters. For the convolution kernel of C channels, 7x7 contains parameters , and the number of 3 3x3 parameters is greatly reduced.

Q2: The role of 1x1 convolution kernel

Answer 2

Increase the nonlinearity of the model without affecting the receptive field 1x1 winding machine is equivalent to linear transformation, and the non-linear activation function plays a non-linear role

Q3: The effect of network depth on results (in the same year, Google also independently released the network GoogleNet with a depth of 22 layers)

Answer 3

VGG and GoogleNet models are deep Small convolution VGG only uses 3x3, while GoogleNet uses 1x1, 3x3, 5x5, the model is more complicated (the model began to use a large convolution kernel to reduce the calculation of the subsequent machine layer)

Implement On Keras

In [3]:

```
#Importing library
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D,MaxPool2
from keras.layers.normalization import BatchNormalization
import numpy as np

np.random.seed(1000)
```

In [4]:

```
model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same", activ
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))


model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))


model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))


model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Flatten())
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=2, activation="softmax"))
```

In [6]:

```
print("The output of this will be the summary of the model which I just created.")
model.summary()
```

```
The output of this will be the summary of the model which I just created.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 224, 224, 64)      1792
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 224, 224, 64)      36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 112, 112, 128)     73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 112, 112, 128)     147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 128)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 56, 56, 256)       295168
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 56, 56, 256)       590080
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 56, 56, 256)       590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 256)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 512)       1180160
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 28, 28, 512)       2359808
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 28, 28, 512)       2359808
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 512)       0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 4096)              102764544
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 8194
=================================================================
Total params: 134,268,738
Trainable params: 134,268,738
Non-trainable params: 0
_________________________________________________________________
```

Here I have started with initialising the model by specifying that the model is a sequential model. After

initialising the model I add

→ 2 x convolution layer of 64 channel of 3x3 kernal and same padding

→ 1 x maxpool layer of 2x2 pool size and stride 2x2

→ 2 x convolution layer of 128 channel of 3x3 kernal and same padding

→ 1 x maxpool layer of 2x2 pool size and stride 2x2

→ 3 x convolution layer of 256 channel of 3x3 kernal and same padding

→ 1 x maxpool layer of 2x2 pool size and stride 2x2

→ 3 x convolution layer of 512 channel of 3x3 kernal and same padding

→ 1 x maxpool layer of 2x2 pool size and stride 2x2

→ 3 x convolution layer of 512 channel of 3x3 kernal and same padding

→ 1 x maxpool layer of 2x2 pool size and stride 2x2

I also add relu(Rectified Linear Unit) activation to each layers so that all the negative values are not passed to the next layer.

After creating all the convolution I pass the data to the dense layer so for that I flatten the vector which comes out of the convolutions and add

→ 1 x Dense layer of 4096 units

→ 1 x Dense layer of 4096 units

→ 1 x Dense Softmax layer of 2 units

I will use RELU activation for both the dense layer of 4096 units so that I stop forwarding negative values through the network. I use a 2 unit dense layer in the end with softmax activation as I have 2 classes to predict from in the end which are dog and cat. The softmax layer will output the value between 0 and 1 based on the confidence of the model that which class the images belongs to.

After the creation of softmax layer the model is finally prepared.

In [ ]: