VGGNet (Visual Geometry Group Network)

VGGNet (Visual Geometry Group Network) is a convolutional neural network architecture proposed by the Visual Geometry Group at Oxford University. It achieved excellent performance on the ImageNet Large Scale Visual Recognition Challenge in 2014.

The architecture consists of 16 layers and is divided into 5 blocks, each containing convolutional layers followed by max-pooling layers. The layers are named according to the block they belong to and their order within the block.

Layer-by-layer explanation of the VGGNet architecture:

Block 1

Input: 224 x 224 x 3 image Convolutional layer with 64 filters, 3 x 3 kernel size, and ReLU activation Convolutional layer with 64 filters, 3 x 3 kernel size, and ReLU activation Max pooling layer with a 2 x 2 window and stride of 2

Block 2

Convolutional layer with 128 filters, 3 x 3 kernel size, and ReLU activation Convolutional layer with 128 filters, 3 x 3 kernel size, and ReLU activation Max pooling layer with a 2 x 2 window and stride of 2

Block 3

Convolutional layer with 256 filters, 3 x 3 kernel size, and ReLU activation Convolutional layer with 256 filters, 3 x 3 kernel size, and ReLU activation Convolutional layer with 256 filters, 3 x 3 kernel size, and ReLU activation Max pooling layer with a 2 x 2 window and stride of 2

Block 4

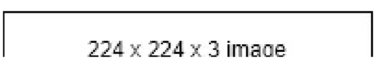
Convolutional layer with 512 filters, 3 x 3 kernel size, and ReLU activation Convolutional layer with 512 filters, 3 x 3 kernel size, and ReLU activation Convolutional layer with 512 filters, 3 x 3 kernel size, and ReLU activation Max pooling layer with a 2 x 2 window and stride of 2

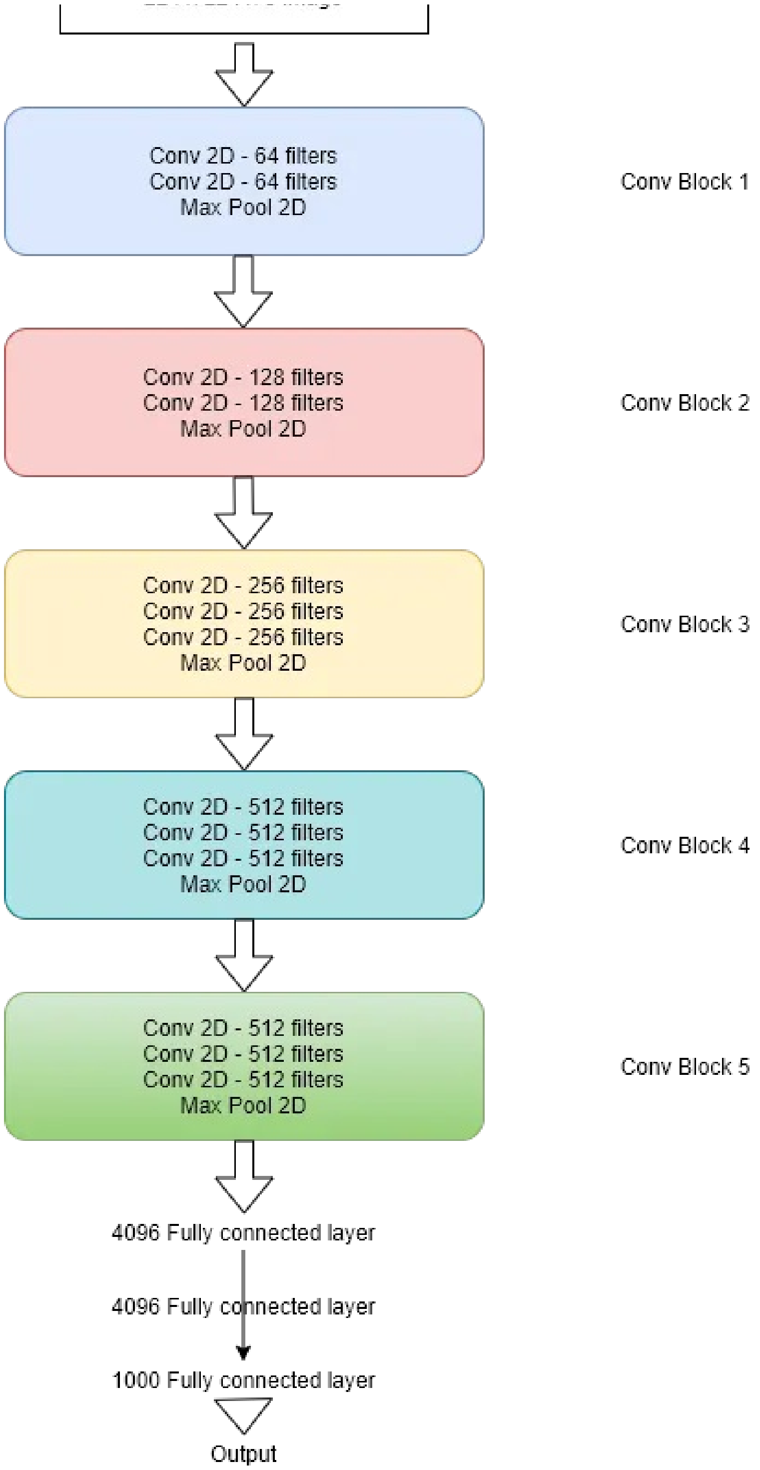
Block 5

Convolutional layer with 512 filters, 3 x 3 kernel size, and ReLU activation Convolutional layer with 512 filters, 3 x 3 kernel size, and ReLU activation Convolutional layer with 512 filters, 3 x 3 kernel size, and ReLU activation Max pooling layer with a 2 x 2 window and stride of 2

Output

1. **Flatten layer**
2. **Fully connected layer with 4096 neurons and ReLU activation**
3. **Dropout layer with 0.5 dropout rate**
4. **Fully connected layer with 4096 neurons and ReLU activation**
5. **Dropout layer with 0.5 dropout rate**
6. **Fully connected output layer with 1000 neurons and Softmax activation**





VGG paper link — https://arxiv.org/abs/1409.1556

VGG 16 architecture and implementation using Tensorflow:

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256

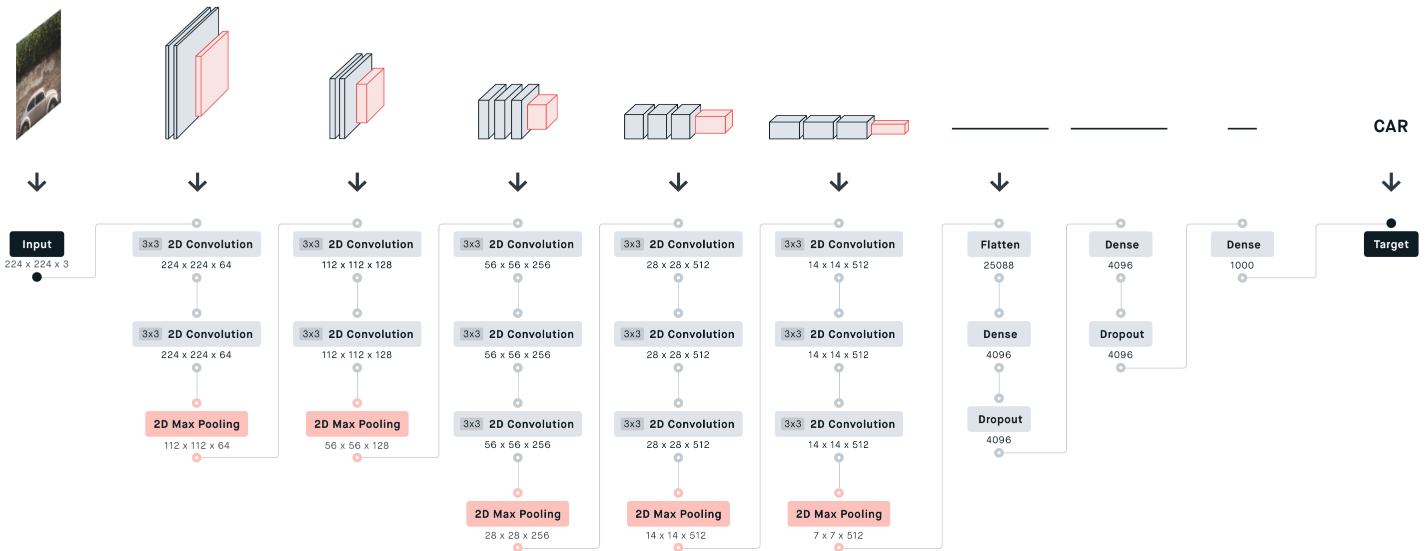
					conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

The above figure shows all the VGG architectures. The architecture of VGG 16 is highlighted in red. A simpler version of the architecture is presented.

VGG network uses Max Pooling and ReLU activation function. All the hidden layers use ReLU activation and the last Dense layer uses Softmax activation. MaxPooling is performed over a 2x2 pixel window with a stride of 2.

VGG 16 has 5 convolutional blocks and 3 fully connected layers. Each block consists of 2 or more Convolutional layers and a Max Pool layer.

Architectural Flow



Importing the libraries

In [ ]:

```
# import necessary layers

from tensorflow.keras.layers import Input, Conv2D
from tensorflow.keras.layers import MaxPool2D, Flatten, Dense
from tensorflow.keras import Model
```

Input:

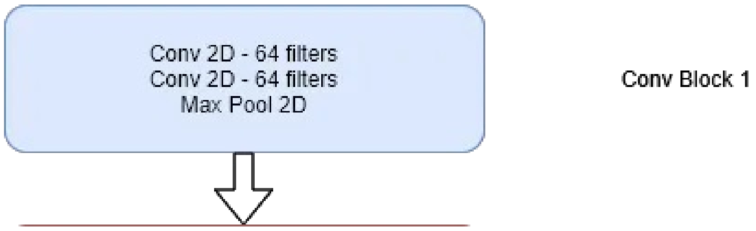
In [ ]:

```python
# input
input = Input(shape =(224,224,3))
```

Input is a 224x224 RGB image, so 3 channels.

Conv Block 1:

It has two Conv layers with 64 filters each, followed by Max Pooling.

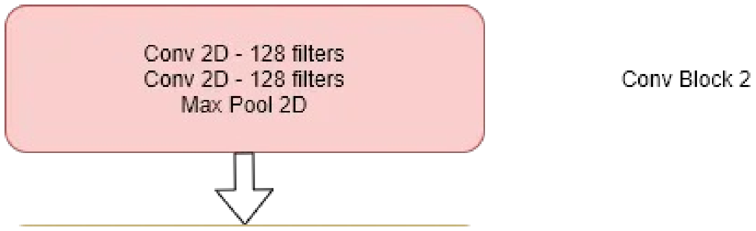


In [ ]:

```python
# 1st Conv Block

x = Conv2D (filters =64, kernel_size =3, padding ='same', activation='relu')(input)
x = Conv2D (filters =64, kernel_size =3, padding ='same', activation='relu')(x)
x = MaxPool2D(pool_size =2, strides =2, padding ='same')(x)
```

Conv Block 2:

It has two Conv layers with 128 filters followed by Max Pooling.

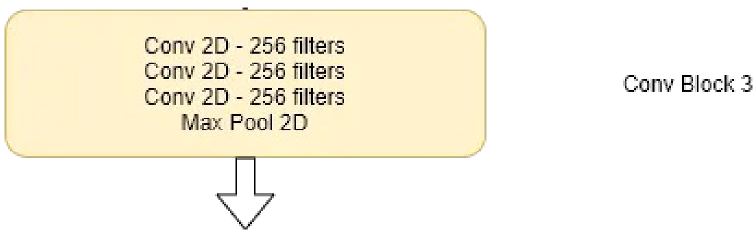


In [ ]:

```python
# 2nd Conv Block

x = Conv2D (filters =128, kernel_size =3, padding ='same', activation='relu')(x)
x = Conv2D (filters =128, kernel_size =3, padding ='same', activation='relu')(x)
x = MaxPool2D(pool_size =2, strides =2, padding ='same')(x)
```

Conv Block 3:

It has three Conv layers with 256 filters followed by Max Pooling.

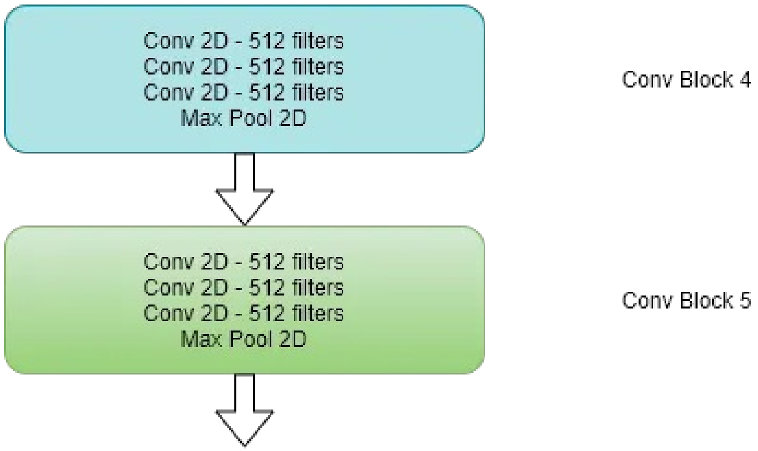


In [ ]:

```python
# 3rd Conv block

x = Conv2D (filters =256, kernel_size =3, padding ='same', activation='relu')(x)
x = Conv2D (filters =256, kernel_size =3, padding ='same', activation='relu')(x)
x = Conv2D (filters =256, kernel_size =3, padding ='same', activation='relu')(x)
x = MaxPool2D(pool_size =2, strides =2, padding ='same')(x)
```

Conv Block 4 and 5:

Both Conv blocks 4 and 5 have 3 Conv layers with 512 filters followed by Max Pooling.

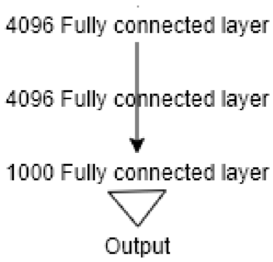


In [ ]:

```
# 4th Conv block

x = Conv2D (filters =512, kernel_size =3, padding ='same', activation='relu')(x)
x = Conv2D (filters =512, kernel_size =3, padding ='same', activation='relu')(x)
x = Conv2D (filters =512, kernel_size =3, padding ='same', activation='relu')(x)
x = MaxPool2D(pool_size =2, strides =2, padding ='same')(x)

# 5th Conv block

x = Conv2D (filters =512, kernel_size =3, padding ='same', activation='relu')(x)
x = Conv2D (filters =512, kernel_size =3, padding ='same', activation='relu')(x)
x = Conv2D (filters =512, kernel_size =3, padding ='same', activation='relu')(x)
x = MaxPool2D(pool_size =2, strides =2, padding ='same')(x)
```

Dense layers:

There are 3 fully connected layers, the first two layers with 4096 hidden units and ReLU activation and the last output layer with 1000 hidden units and Softmax activation.



In [ ]:

```
# Fully connected layers
x = Flatten()(x)
x = Dense(units = 4096, activation ='relu')(x)
x = Dense(units = 4096, activation ='relu')(x)
output = Dense(units = 1000, activation ='softmax')(x)
```

Creating the Model:

In [ ]:

```
# creating the model

model = Model (inputs=input, outputs =output)
model.summary()
```

```
Model: "model"
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 conv2d (Conv2D)             (None, 224, 224, 64)      1792

 conv2d_1 (Conv2D)           (None, 224, 224, 64)      36928

 max_pooling2d (MaxPooling2D  (None, 112, 112, 64)     0
 )

 conv2d_2 (Conv2D)           (None, 112, 112, 128)     73856

 conv2d_3 (Conv2D)           (None, 112, 112, 128)     147584

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 128)      0
 2D)

 conv2d_4 (Conv2D)           (None, 56, 56, 256)       295168

 conv2d_5 (Conv2D)           (None, 56, 56, 256)       590080

 conv2d_6 (Conv2D)           (None, 56, 56, 256)       590080

 max_pooling2d_2 (MaxPooling  (None, 28, 28, 256)      0
 2D)

 conv2d_7 (Conv2D)           (None, 28, 28, 512)       1180160

 conv2d_8 (Conv2D)           (None, 28, 28, 512)       2359808

 conv2d_9 (Conv2D)           (None, 28, 28, 512)       2359808

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 512)      0
 2D)

 conv2d_10 (Conv2D)          (None, 14, 14, 512)       2359808

 conv2d_11 (Conv2D)          (None, 14, 14, 512)       2359808

 conv2d_12 (Conv2D)          (None, 14, 14, 512)       2359808

 max_pooling2d_4 (MaxPooling  (None, 7, 7, 512)        0
 2D)

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 4096)              102764544

 dense_1 (Dense)             (None, 4096)              16781312

 dense_2 (Dense)             (None, 1000)              4097000

=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

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

In [ ]:

```
# plotting the model

from tensorflow.keras.utils import model_to_dot
from IPython.display import SVG
import pydot
import graphviz

SVG(model_to_dot(model, show_shapes=True, show_layer_names=True, rankdir='TB',expand_nes
ted=False, dpi=60, subgraph=False).create(prog='dot',format='svg'))
```

Out[ ]:

Conclusion

The VGG network is a very simple Convolutional Neural Network, and due to its simplicity is very easy to implement using Tensorflow. It has only Conv2D, MaxPooling, and Dense layers. VGG 16 has a total of 138 million trainable parameters.

Also, VGG-16 is a popular deep neural network architecture that has been used in many computer vision tasks, such as image classification, object detection, and segmentation. It consists of 16 convolutional and fully connected layers, with the convolutional layers using small kernel sizes (3x3) and a large number of filters to extract rich features from the input image. VGG-16 has achieved state-of-the-art performance on various image classification benchmarks, and its success has influenced the development of other deep neural network architectures.

Author: Shobhandeb Paul
Github: https://github.com/herbert0419
Linkedin: https://www.linkedin.com/in/shobhandeb-paul-b6914a168/

In [ ]: