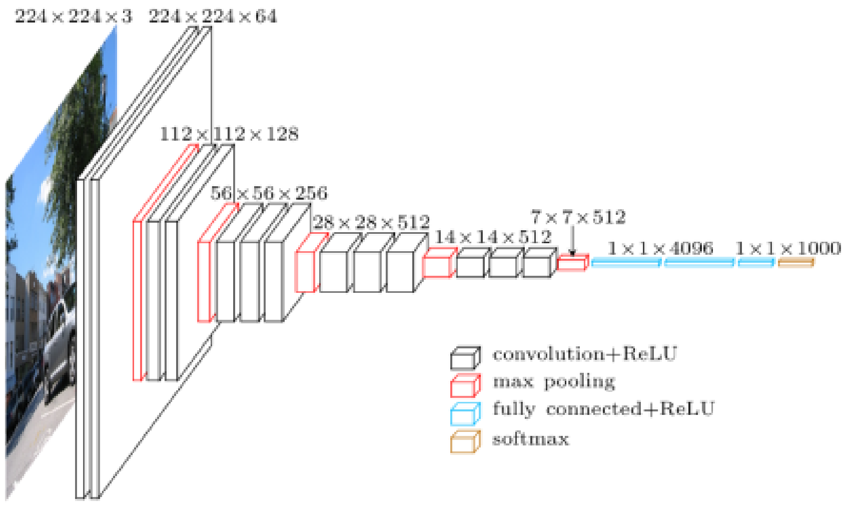
VGG Architecture (CNN)

VGG (Visual Geometry Group) is a family of convolutional neural network (CNN) architectures developed by researchers at the University of Oxford. The VGG models are widely known for their simple and uniform architecture, making them easy to understand and implement.

The key idea behind VGG architectures is to use a series of smaller convolutional filters (3x3) with a small stride and always keep the spatial resolution unchanged while increasing the number of filters in deeper layers. This approach proved to be effective and achieved significant performance improvements on various computer vision tasks.



There are several versions of the VGG architecture, named according to the number of layers they have. The most famous variants are VGG16 and VGG19:

VGG11: This version has 11 layers, including 8 convolutional layers followed by 3 fully connected layers.

VGG13: Similar to VGG11, but with 13 layers, including 10 convolutional layers and 3 fully connected layers.

VGG16: This version has 16 layers, comprising 13 convolutional layers and 3 fully connected layers. It is deeper than VGG11 and VGG13, allowing it to capture more complex patterns and features.

VGG19: VGG19 is the most extensive version with 19 layers. It consists of 16 convolutional layers and 3 fully connected layers. Like VGG16, it can capture even more intricate features and patterns.

The naming convention for VGG architectures indicates the number of layers in the network. For example, VGG16 has 16 layers, VGG19 has 19 layers, VGG13 has 13 layers and VGG11 has 11 layers.

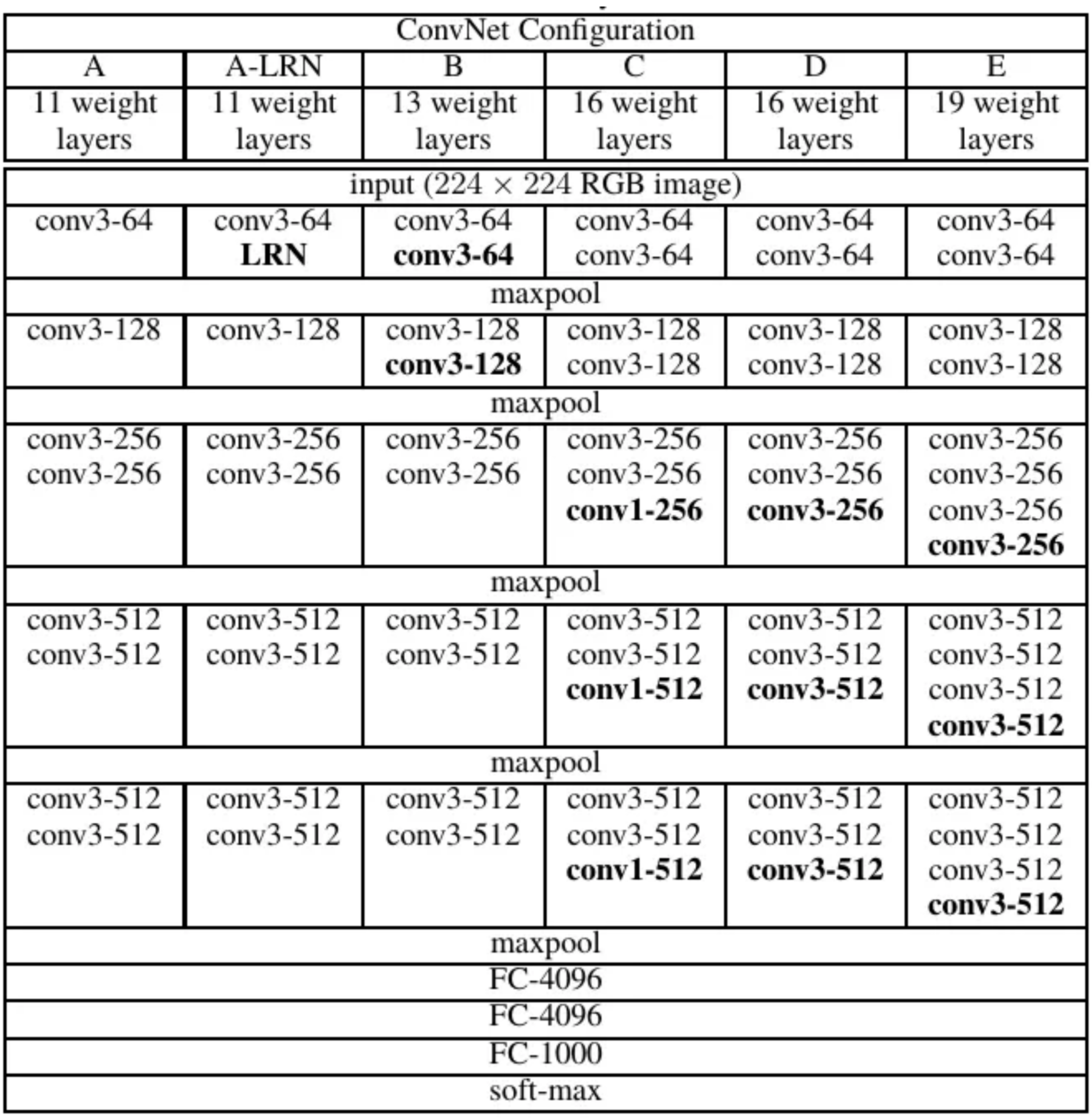
Architecture Configuration :

The below figure contains the Convolution Neural Network configuration of the VGG net with the

following layers:

- `VGG-11`
- `VGG-11 (LRN)`
- `VGG-13`

- `VGG-16 (Conv1)`
- `VGG-16`
- `VGG-19`



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					

Both VGG-16 and VGG-19 have been widely used and contributed significantly to the development and understanding of deep learning. However, due to their large number of parameters, they can be computationally expensive to train and deploy.

The VGG architectures have been influential in the field of computer vision and served as the basis for various other CNN architectures that followed. However, due to their depth, VGG models can be computationally expensive and have relatively high memory requirements, which has led to the development of more efficient architectures like ResNet, DenseNet, and MobileNet.

It's worth noting that since my knowledge is up to September 2021, there might have been additional developments or variations of the VGG architecture beyond that date. Always check the latest research and literature for the most up-to-date information.

VGG-11 Architecture

As of my last update in September 2021, VGG11 refers to a specific variant of the VGG (Visual Geometry Group) convolutional neural network architecture. The VGG network was introduced by the Visual Geometry Group at the University of Oxford and has become a classic and influential architecture in the field of deep learning.

The VGG11 architecture has 11 layers, including 8 convolutional layers and 3 fully connected layers. Each convolutional layer is followed by a max-pooling layer for down-sampling. The architecture is characterized by using small 3x3 convolutional filters with a stride of 1, which helps in learning hierarchical features from the input image.

Here's a summary of the VGG11 architecture:

1. Convolutional Layer: 64 filters, 3x3 kernel, ReLU activation
2. Max-Pooling Layer: 2x2 window, stride 2
3. Convolutional Layer: 128 filters, 3x3 kernel, ReLU activation
4. Max-Pooling Layer: 2x2 window, stride 2
5. Convolutional Layer: 256 filters, 3x3 kernel, ReLU activation
6. Convolutional Layer: 256 filters, 3x3 kernel, ReLU activation
7. Max-Pooling Layer: 2x2 window, stride 2
8. Convolutional Layer: 512 filters, 3x3 kernel, ReLU activation
9. Convolutional Layer: 512 filters, 3x3 kernel, ReLU activation
10. Max-Pooling Layer: 2x2 window, stride 2
11. Fully Connected Layer: 4096 units, ReLU activation
12. Fully Connected Layer: 4096 units, ReLU activation
13. Fully Connected Layer: 1000 units (for ImageNet classification)

Note that the last fully connected layer has 1000 units for ImageNet classification since the VGG network was originally trained on the ImageNet dataset, which contains 1000 classes.

Researchers have also developed other variants like VGG16 and VGG19, which have more layers and increased model capacity, but VGG11 is the simpler and more lightweight version. It's important to keep in mind that newer architectures might have been introduced after my last update. Always refer to the latest research papers and documentation for the most up-to-date information.

In [ ]:

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

def build_vgg11(input_shape=(224, 224, 3), num_classes=1000):
    model = Sequential()

    # Block 1
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=input_shape))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 2
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 3
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 4
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 5
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Fully connected layers
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))

    return model

   #Create VGG11 model
vgg11_model = build_vgg11()

   #Optional: Print model summary
vgg11_model.summary()
```

VGG-13 Architecture

As of my last update in September 2021, there is only one official version of the VGG-13 architecture. VGG (Visual Geometry Group) is a convolutional neural network (CNN) architecture developed by the Visual Geometry Group at the University of Oxford. The "13" in VGG-13 refers to the number of weight layers in the network.

The VGG-13 architecture is part of the VGG family, which includes several other architectures such as VGG-16 and VGG-19. The primary difference between these architectures lies in the number of layers. The more layers, the deeper the network, but it also means more parameters and increased computational complexity.

Here is the basic structure of the VGG-13 architecture:

1. Input (224x224x3 image)
2. Convolutional Layers (with ReLU activation and 3x3 filters):
   - 2 x 64 filters
   - 2 x 128 filters
   - 3 x 256 filters
   - 3 x 512 filters
   - 3 x 512 filters
3. Max Pooling Layers (2x2)
4. Fully Connected Layers:
   - 1 x 4096 neurons
   - 1 x 4096 neurons
   - 1 x 1000 neurons (output layer for ImageNet's 1000 classes)

Note that the number of output neurons in the last fully connected layer is 1000 because VGG was originally designed for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition, which has 1000 different object classes.

If there have been any updates or additional versions released after September 2021, I wouldn't be aware of them. It's always a good idea to refer to the latest research papers or official repositories for the most up-to-date information.

VGG is known for its simplicity and effectiveness in image classification tasks. The number in the architecture's name refers to the number of layers in the network. VGG-13 has 13 layers, including convolutional layers, max-pooling layers, and fully connected layers. Here is the architecture of VGG-13:

1. Input (224x224x3) - The network takes an input image of size 224x224 pixels with three color channels (RGB).
2. Convolutional Layer (64 filters, 3x3 kernels, stride 1, padding 1) , ReLU Activation
3. Convolutional Layer (64 filters, 3x3 kernels, stride 1, padding 1) , ReLU Activation
4. Max-Pooling Layer (2x2 pool size, stride 2) - A max-pooling operation with a pool size of 2x2 and a stride of 2 is used to downsample the spatial dimensions.
5. Convolutional Layer (128 filters, 3x3 kernels, stride 1, padding 1) , ReLU Activation
6. Convolutional Layer (128 filters, 3x3 kernels, stride 1, padding 1) , ReLU Activation
7. Max-Pooling Layer (2x2 pool size, stride 2) - Another max-pooling layer with a pool size of 2x2 and a stride of 2 is used for downsampling.
8. Convolutional Layer (256 filters, 3x3 kernels, stride 1, padding 1) , ReLU Activation
9. Convolutional Layer (256 filters, 3x3 kernels, stride 1, padding 1) , ReLU Activation
10. Max-Pooling Layer (2x2 pool size, stride 2) - Another max-pooling layer with a pool size of 2x2 and a stride of 2 is used for downsampling.
11. Convolutional Layer (512 filters, 3x3 kernels, stride 1, padding 1) , ReLU Activation
12. Convolutional Layer (512 filters, 3x3 kernels, stride 1, padding 1) , ReLU Activation
13. Max-Pooling Layer (2x2 pool size, stride 2) - Another max-pooling layer with a pool size of 2x2 and a stride of 2 is used for downsampling.
14. Convolutional Layer (512 filters, 3x3 kernels, stride 1, padding 1) , ReLU Activation
15. Convolutional Layer (512 filters, 3x3 kernels, stride 1, padding 1) , ReLU Activation
16. Max-Pooling Layer (2x2 pool size, stride 2) - Another max-pooling layer with a pool size of 2x2 and a stride of 2 is used for downsampling.
17. Fully Connected Layer (4096 units) - A fully connected layer with 4096 units is used for high-level feature representation , Followed by ReLU Activation
18. Fully Connected Layer (4096 units) - Another fully connected layer with 4096 units is applied , Followed by ReLU Activation
19. Output Layer (1000 units) - The final fully connected layer with 1000 units represents the output classes (in the original VGG-13 implementation, it was trained on the ImageNet dataset with 1000 classes).
20. Softmax Activation - The softmax activation function is applied to produce class probabilities.

Note: The original VGG-13 model is quite deep, making it computationally expensive and prone to overfitting on smaller datasets. Other variants of the VGG architecture, such as VGG-16 and VGG-19, also exist and have even more layers. For many practical applications, pre-trained versions of VGG models (typically trained on ImageNet) are commonly used for transfer learning on various computer vision tasks.

In [ ]:
```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

def VGG13(input_shape, num_classes):
    model = Sequential()
    # Block 1
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=input_shape))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 2
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 3
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 4
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 5
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Fully Connected layers
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))

    return model

    #Example usage

input_shape = (224, 224, 3)  # Input shape of your images

num_classes = 1000  # Number of classes in your dataset

model = VGG13(input_shape, num_classes)
```

VGG-16 Architecture

VGG16 is a convolutional neural network (CNN) architecture that was proposed by the Visual Geometry Group (VGG) at the University of Oxford. It was introduced in the paper titled "Very Deep Convolutional Networks for Large-Scale Image Recognition" by Karen Simonyan and Andrew Zisserman in 2014. VGG16 is a variant of the VGG network, and it gained popularity for its simplicity and outstanding performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014.

The key characteristic of VGG16 is its depth, as it consists of 16 layers (hence the name "VGG16"). The architecture is designed to have a homogeneous structure, using 3x3 convolutional layers and 2x2 max-pooling layers throughout the network. Here's a summary of the VGG16 architecture:

1. Input layer: The input to VGG16 is an RGB image of size 224x224 pixels.

2. Convolutional layers: The network consists of 13 convolutional layers, each followed by a Rectified Linear Unit (ReLU) activation function. The number of filters increases with the depth of the network, starting with 64 filters in the first layer and doubling in each subsequent layer, reaching 512 filters in the deepest layers.

3. Max-pooling layers: After every two convolutional layers, there is a max-pooling layer with a 2x2 window and a stride of 2. Max-pooling helps reduce the spatial dimensions and control the number of parameters in the network.

4. Fully connected layers: Once the convolutional and pooling layers are done, VGG16 has three fully connected layers with 4096 units each. The output of these layers goes through ReLU activations as well.

5. Output layer: The final layer is a softmax activation layer with 1000 units (for the 1000 classes in the ImageNet dataset).

The VGG16 architecture is simple, with small 3x3 filters and max-pooling layers that allow for a more extensive exploration of the image's spatial information. However, the main drawback of VGG16 is its high number of parameters, which makes it computationally expensive and memory-intensive. Despite this, VGG16 has been an essential milestone in the development of deep learning architectures and has served as a baseline for many subsequent CNN models.

VGG16 is a popular deep convolutional neural network (CNN) architecture developed by the Visual Geometry Group (VGG) at the University of Oxford. It was introduced in the paper titled "Very Deep Convolutional Networks for Large-Scale Image Recognition" by Karen Simonyan and Andrew Zisserman in 2014. VGG16 is part of a series of VGG models that vary in depth, with the number 16 referring to the number of weight layers in the network.

Here is the detailed architecture of VGG16:

1. Input Layer:
   - The input to the network is a color image of fixed size (224x224 pixels).
2. Convolutional Layers:
   - The network consists of 13 convolutional layers stacked one after the other.
   - Each convolutional layer uses small 3x3 filters with a stride of 1.
   - Padding is used to keep the spatial dimensions the same after each convolution (pad=1).
   - The number of filters in each layer increases progressively, starting with 64 in the first layer and doubling after every two layers. The number of filters in each layer is as follows:
     - 64 filters in 1st and 2nd convolutional layers.
     - 128 filters in 3rd and 4th convolutional layers.
     - 256 filters in 5th and 6th convolutional layers.
     - 512 filters in 7th, 8th, and 9th convolutional layers.
     - 512 filters in 10th, 11th, and 12th convolutional layers.
3. Activation Function:
   - After each convolutional layer, a rectified linear unit (ReLU) activation function is applied to introduce non-linearity.
4. Max Pooling Layers:
   - After each set of two convolutional layers, a max pooling layer is applied to reduce spatial dimensions.
   - The max pooling layers have a 2x2 window size with a stride of 2, resulting in a halving of spatial dimensions.
5. Fully Connected Layers (Dense Layers):
   - VGG16 has three fully connected layers at the end.
   - Each fully connected layer consists of 4096 neurons.
   - ReLU activation is used for these fully connected layers as well.
6. Output Layer:

- The final output layer consists of 1000 neurons (for the 1000 classes in the ImageNet dataset used for training).
- A softmax activation function is applied to produce probabilities for each class.

The total number of trainable parameters in VGG16 is approximately 138 million, making it a large and computationally expensive model. While VGG16 has been surpassed in performance by more modern architectures like ResNet, it played a significant role in advancing the field of deep learning and remains a valuable reference in computer vision research.

The main characteristic of VGG16 is its simplicity and uniformity in design. It uses small 3x3 convolutional filters throughout the network, which allows for a deeper architecture while keeping the number of parameters manageable. The network follows a pattern of stacking multiple convolutional layers, followed by max-pooling layers to downsample the spatial dimensions, and then followed by fully connected layers for classification.

Here is the detailed process architecture of VGG16:

1. Input: The input image size for VGG16 is typically 224x224x3 (RGB image with a resolution of 224x224).

2. Convolutional Layers:

   - Conv1: 64 filters, 3x3 kernel, stride 1, padding 1, ReLU activation
   - Conv2: 64 filters, 3x3 kernel, stride 1, padding 1, ReLU activation
   - MaxPool1: 2x2 max-pooling, stride 2

   - Conv3: 128 filters, 3x3 kernel, stride 1, padding 1, ReLU activation

   - Conv4: 128 filters, 3x3 kernel, stride 1, padding 1, ReLU activation
   - MaxPool2: 2x2 max-pooling, stride 2

   - Conv5: 256 filters, 3x3 kernel, stride 1, padding 1, ReLU activation

   - Conv6: 256 filters, 3x3 kernel, stride 1, padding 1, ReLU activation
   - Conv7: 256 filters, 3x3 kernel, stride 1, padding 1, ReLU activation
   - MaxPool3: 2x2 max-pooling, stride 2

   - Conv8: 512 filters, 3x3 kernel, stride 1, padding 1, ReLU activation

   - Conv9: 512 filters, 3x3 kernel, stride 1, padding 1, ReLU activation
   - Conv10: 512 filters, 3x3 kernel, stride 1, padding 1, ReLU activation
   - MaxPool4: 2x2 max-pooling, stride 2

   - Conv11: 512 filters, 3x3 kernel, stride 1, padding 1, ReLU activation

   - Conv12: 512 filters, 3x3 kernel, stride 1, padding 1, ReLU activation
   - Conv13: 512 filters, 3x3 kernel, stride 1, padding 1, ReLU activation
   - MaxPool5: 2x2 max-pooling, stride 2
3. Fully Connected Layers:

   - FC1: 4096 units, ReLU activation
   - FC2: 4096 units, ReLU activation
   - FC3 (Output layer): 1000 units (corresponding to the number of classes in ImageNet), softmax activation for classification.

Please note that the output layer with 1000 units represents the 1000 classes in the ImageNet dataset. For other applications, the number of units in the output layer would be adjusted based on the specific classification task.

The total number of trainable parameters in VGG16 is approximately 138 million, making it a relatively large model. VGG16, despite its age, has served as the basis for many subsequent CNN architectures and has significantly influenced the development of deep learning in computer vision.

In [ ]:
```python
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

def VGG16():
    model = Sequential()

    # Block 1
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(224, 224, 3)))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 2
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 3
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 4
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 5
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Fully Connected Layers
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(1000, activation='softmax'))  # 1000 classes for ImageNet

    return model

    #Create the VGG16 model
vgg16_model = VGG16()
```

VGG-19 Architecture

VGG19 is a deep convolutional neural network architecture that was proposed by researchers at the Visual Geometry Group (VGG) at the University of Oxford. It is an extension of the original VGG16 architecture, with 19 layers (hence the name VGG19). The network is known for its simplicity and uniformity, with small 3x3 convolutional filters and max-pooling layers throughout the model. Here's the detailed architecture of VGG19:

1. Input layer: The input to the network is typically an RGB image with a fixed size, such as 224x224 pixels.

2. Convolutional layers:

   - Convolutional layer 1: 64 filters with a kernel size of 3x3, followed by a Rectified Linear Unit (ReLU) activation function.
   - Convolutional layer 2: 64 filters with a kernel size of 3x3, followed by a ReLU activation function.

- Max-pooling layer 1: 2x2 max-pooling with stride 2 to downsample the spatial dimensions.
- Convolutional layer 3: 128 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Convolutional layer 4: 128 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Max-pooling layer 2: 2x2 max-pooling with stride 2.
- Convolutional layer 5: 256 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Convolutional layer 6: 256 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Convolutional layer 7: 256 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Convolutional layer 8: 256 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Max-pooling layer 3: 2x2 max-pooling with stride 2.
- Convolutional layer 9: 512 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Convolutional layer 10: 512 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Convolutional layer 11: 512 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Convolutional layer 12: 512 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Max-pooling layer 4: 2x2 max-pooling with stride 2.
- Convolutional layer 13: 512 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Convolutional layer 14: 512 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Convolutional layer 15: 512 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Convolutional layer 16: 512 filters with a kernel size of 3x3, followed by a ReLU activation function.
- Max-pooling layer 5: 2x2 max-pooling with stride 2.

3. Fully Connected layers:
   - Fully Connected layer 1: 4096 neurons with a ReLU activation function.
   - Fully Connected layer 2: 4096 neurons with a ReLU activation function.
   - Fully Connected layer 3: 1000 neurons with a Softmax activation function (used for classification into 1000 categories in the original ImageNet challenge).
4. Output layer: The output layer depends on the specific task. In the ImageNet challenge, it had 1000 neurons corresponding to the 1000 ImageNet classes.

VGG19 is a powerful architecture and was widely used in various computer vision tasks, including image classification and feature extraction, even though it has more parameters than many contemporary models. It played a crucial role in advancing the field of deep learning and CNN architectures.

In [ ]:
```python
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

def VGG19():
    model = Sequential()

    # Block 1
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(224, 224, 3)))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 2
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 3
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 4
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 5
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Classification block
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(1000, activation='softmax'))  # Assuming 1000 classes for ImageNet dataset

    return model

    #Create the VGG19 model
vgg19_model = VGG19()
```

Advantages of VGG :

1. VGG brings with it a variety of structures based on the same idea. This provides us additional alternatives when it comes to determining which architecture would work best for our application.

2. Non-linearity increased as the number of layers with smaller kernels increased, which is always a good thing in deep learning.

3. VGG resulted in a significant increase in accuracy as well as a significant increase in speed. This was mostly due to the model's increased depth and the addition of pretrained models.

Disadvantages of VGG :

1. This model suffers from the vanishing gradient problem, which I discovered to be a significant drawback. When we look at my validation loss graph, we can see that it is steadily growing. None of the other models were in the same boat. The ResNet architecture was used to tackle the vanishing gradient problem.
2. The older VGG design is slower than the newer ResNet architecture, which introduced the idea of residual learning, another key accomplishment.

Created by Sanjay Panchal
Follow on LinkedIn