CNN Basics

Table of Contents

1. Introduction to Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as ConvNets. CNNs are used for image recognition and classification, object detection, and many other tasks.

CNNs are inspired by the organization of the animal visual cortex, where individual neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

The CNN architecture is made up of these layers:

- Convolutional Layer (1D or 2D)
- Pooling Layer
- Fully Connected Layer

in this document, we will cover CNN how they start from a convolutional layer with the convolution operation and the filters/kernels, then we will move to talking about the padding and strides, and then we will talk about the pooling layers and the fully connected layers.

We will also cover the most famous CNN architectures like LeNet, AlexNet, VGGNet, GoogLeNet, and ResNet.

We will also cover some good regularization techniques like Batch Normalization and Data Augmentation For CNN.

Finally, we will talk about understanding and visualizing CNNs, and we will cover some research papers and articles.

And Ofc all of this will be covered with code examples in Python using TensorFlow and Keras.

2. Basic Concepts and Building Blocks

Convolution Operation

Filters/Kernels

The convolution operation is the building block of a Convolutional Neural Network. It is the first layer to extract features from an input image. The convolution operation involves a kernel that moves over the input image, performing element-wise multiplication with the input image and then summing up the results to produce a feature map.

But How the Filter/kernels are extracted?

The filters are learned during the training process. The network learns the filters that in traditional algorithms were hand-engineered. The filters are initialized randomly and updated during the training process using backpropagation.

But BackPropagation updates the weights, how the filters are updated?

the filters are composed of weights, and the weights are updated during the training process using backpropagation. The filters are updated to minimize the loss function.

so the best way to look at a filter is to look at it as a feature extractor. using a matrix of weights.

But Where are the biases?

The biases are added after the convolution operation. The biases are added to the output of the convolution operation.

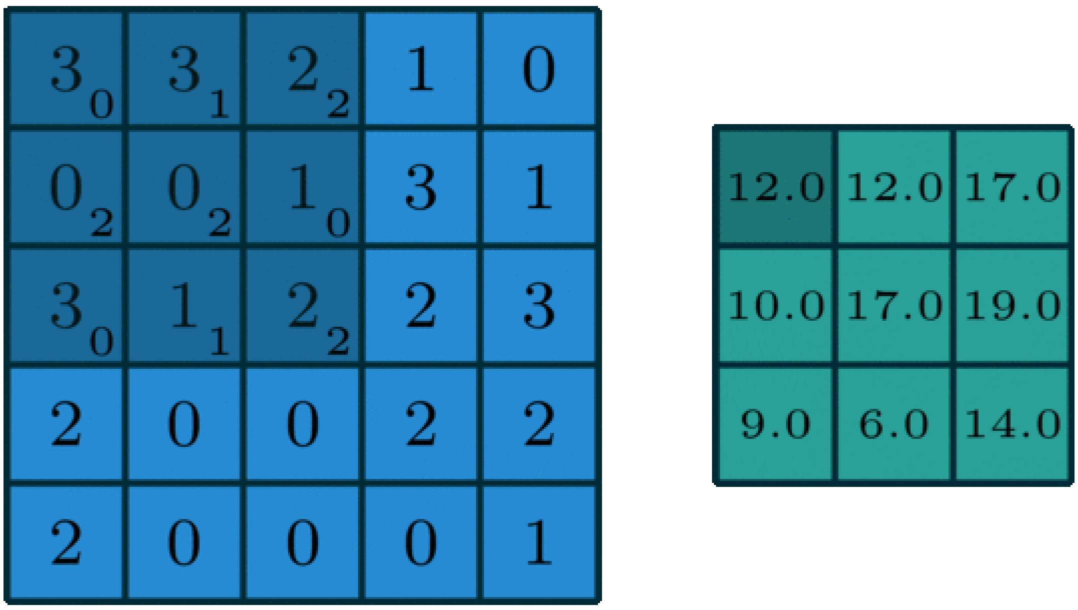

Figure 1 : Convolution operation with a 3x3 kernel

Strides and Padding

The problem with the convolution operation is that the size of the output feature map is smaller than the input feature map. This is because the kernel cannot be placed on the edges of the input feature map. To solve this problem, we can use padding.

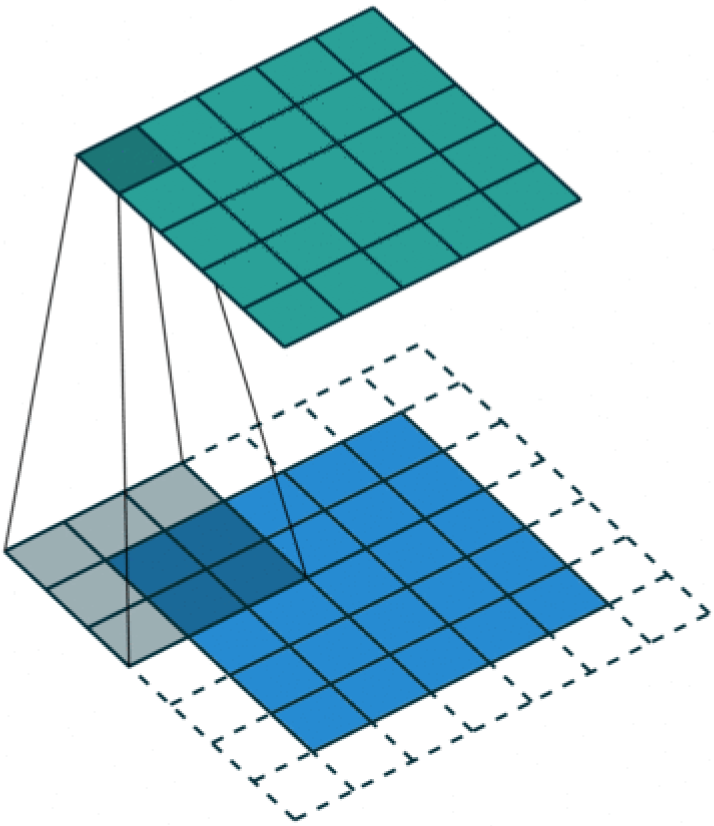

Figure 2 : Padding the input image

Padding is the process of adding zeros to the input image to make the output feature map the same size as the input feature map. Padding can be of two types:

1. Valid Padding: No padding is added to the input image. The output feature map is smaller than the input feature map.
2. Same Padding: Padding is added to the input image to make the output feature map the same size as the input feature map.

The stride is the number of pixels by which the kernel moves after each convolution operation. The stride can be of any size, but a stride of 1 is the most common.

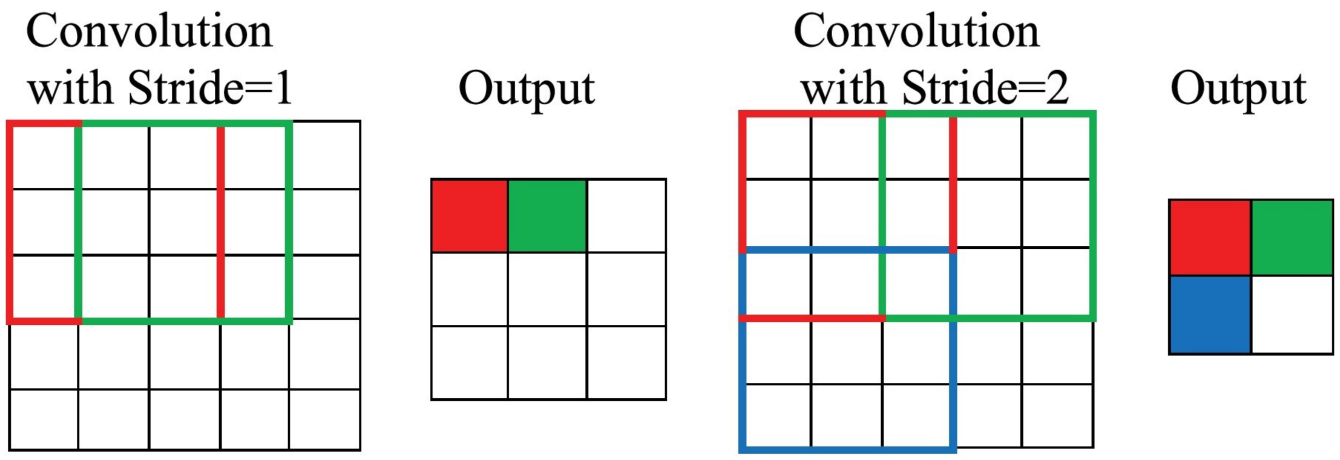

Figure 3 : Stride of 1 vs Stride of 2

Code Examples

```
import tensorflow as tf
from tensorflow.keras import layers

# # Create a convolutional layer
# Padding can be 'valid' or 'same'
# Strides can be any positive integer
# You choose the number of filters and the size of the kernel you want to use
# The activation function is usually 'relu'
conv_layer = layers.Conv2D(filters=16, kernel_size=3, strides=1, padding='same',
activation='relu', input_shape=(28, 28, 1))
```

```
# You would also make a 1D layer by using Conv1D instead of Conv2D
conv_layer = layers.Conv1D(filters=16, kernel_size=3, strides=1, padding='same',
activation='relu', input_shape=(28, 1))
```

Pooling Layers

Pooling layers are used to reduce the spatial dimensions of the input feature map. Pooling layers are used to reduce the number of parameters in the network and to control overfitting.

There are various types of pooling layers:

1. **Max Pooling:** The max pooling operation involves taking the maximum value from the input feature map.

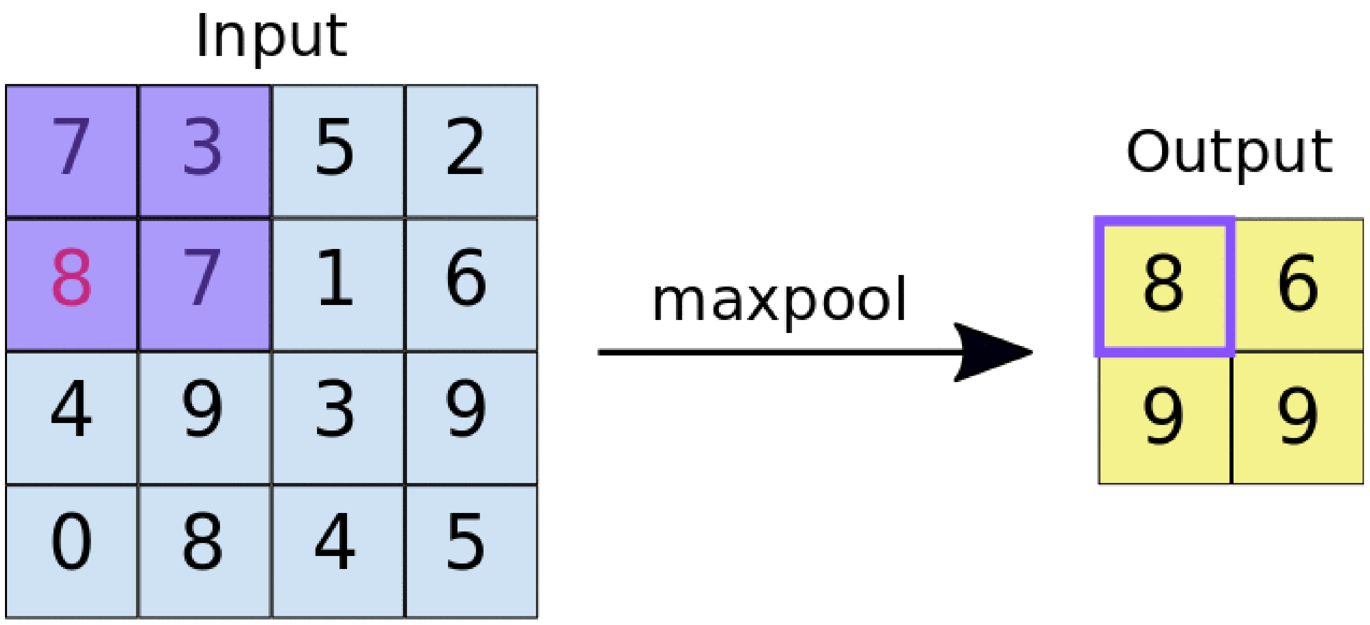

Figure 4 : Max pooling operation

```
# Create a max pooling layer
# Pool size can be any positive integer (2 is 2x2 pooling)
# Strides can be any positive integer
max_pooling_layer = layers.MaxPooling2D(pool_size=2, strides=2)
```

max pooling is used to get the prominent features from the feature map.

2. **Average Pooling:** The average pooling operation involves taking the average value from the input feature map.

```
# Create an average pooling layer
average_pooling_layer = layers.AveragePooling2D(pool_size=2, strides=2)
```

average pooling is used to get the average of the features from the feature map.

3. **Global Average Pooling:** The global average pooling operation involves taking the average value from the entire feature map.

```
# Create a global average pooling layer
global_average_pooling_layer = layers.GlobalAveragePooling2D()
```

global average pooling is used to get the average of the features from the entire feature map.

4. **Global Max Pooling:** The global max pooling operation involves taking the maximum value from the entire feature map.

```
# Create a global max pooling layer
global_max_pooling_layer = layers.GlobalMaxPooling2D()
```

global max pooling is used to get the prominent features from the entire feature map.

Fully Connected Layers

Fully connected layers are used to connect every neuron in one layer to every neuron in another layer. Fully connected layers are used to classify the input image into various classes.

```
# Create a fully connected layer
# The number of units is the number of neurons in the layer
# The activation function is usually 'relu'
fully_connected_layer = layers.Dense(units=128, activation='relu')
output_layer = layers.Dense(units=10, activation='softmax')
```

if we have a 2D input, we need to flatten it before passing it to the fully connected layer.

```
# Flatten the 2D input
flatten_layer = layers.Flatten()
```

3. Architectures of CNNs

LeNet

LeNet is one of the first CNN architectures. It was developed by Yann LeCun in 1998. LeNet was used for handwritten digit recognition.

LeNet consists of the following layers:

1. Convolutional Layer
2. Pooling Layer
3. Convolutional Layer
4. Pooling Layer
5. Fully Connected Layer
6. Fully Connected Layer
7. Output Layer With 10 Units (Number of Classes)

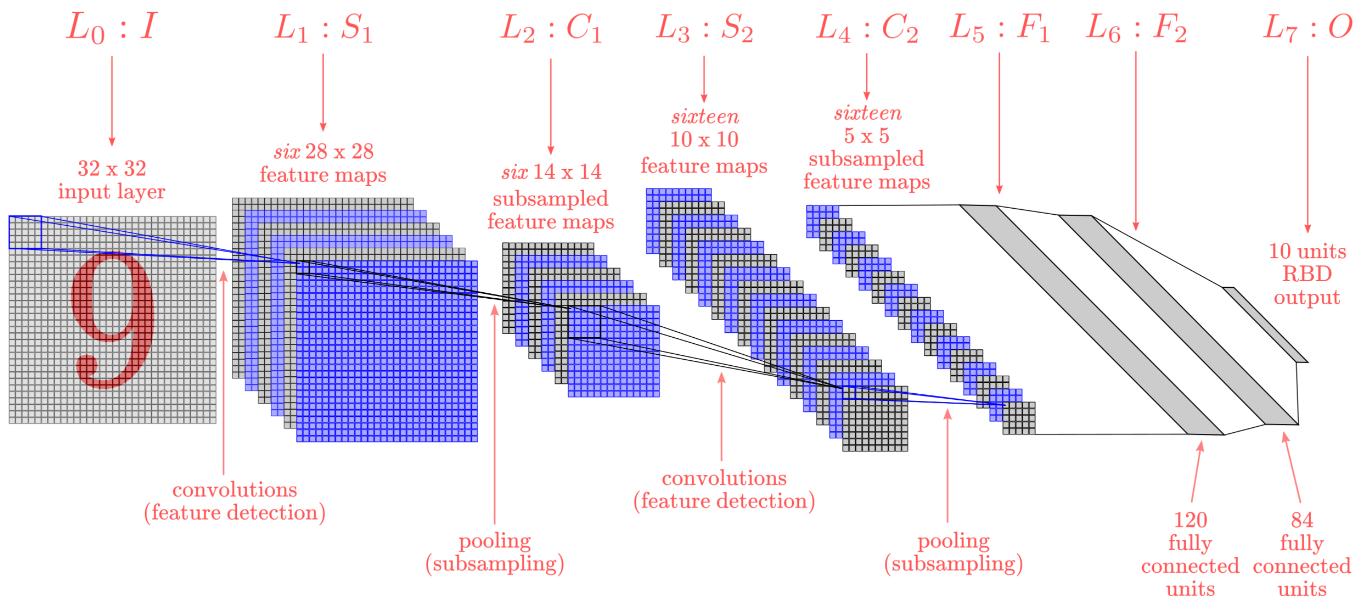
$L_0 : I$
32 x 32 input layer
$L_1 : S_1$
six 28 x 28 feature maps
$L_2 : C_1$
six 14 x 14 subsampled feature maps
$L_3 : S_2$
sixteen 10 x 10 feature maps
$L_4 : C_2$
sixteen 5 x 5 subsampled feature maps
$L_5 : F_1$
$L_6 : F_2$
$L_7 : O$
10 units RBD output
9
convolutions (feature detection)
pooling (subsampling)
convolutions (feature detection)
pooling (subsampling)
120 fully connected units
84 fully connected units

Figure 5 : LeNet architecture

AlexNet

AlexNet is a CNN architecture developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. AlexNet won the ImageNet Large Scale Visual Recognition Challenge in 2012.

AlexNet consists of the following layers:

1. Convolutional Layer
2. Pooling Layer
3. Convolutional Layer
4. Pooling Layer
5. Convolutional Layer
6. Convolutional Layer
7. Convolutional Layer
8. Pooling Layer
9. Fully Connected Layer
10. Fully Connected Layer
11. Output Layer With 1000 Units (Number of Classes)

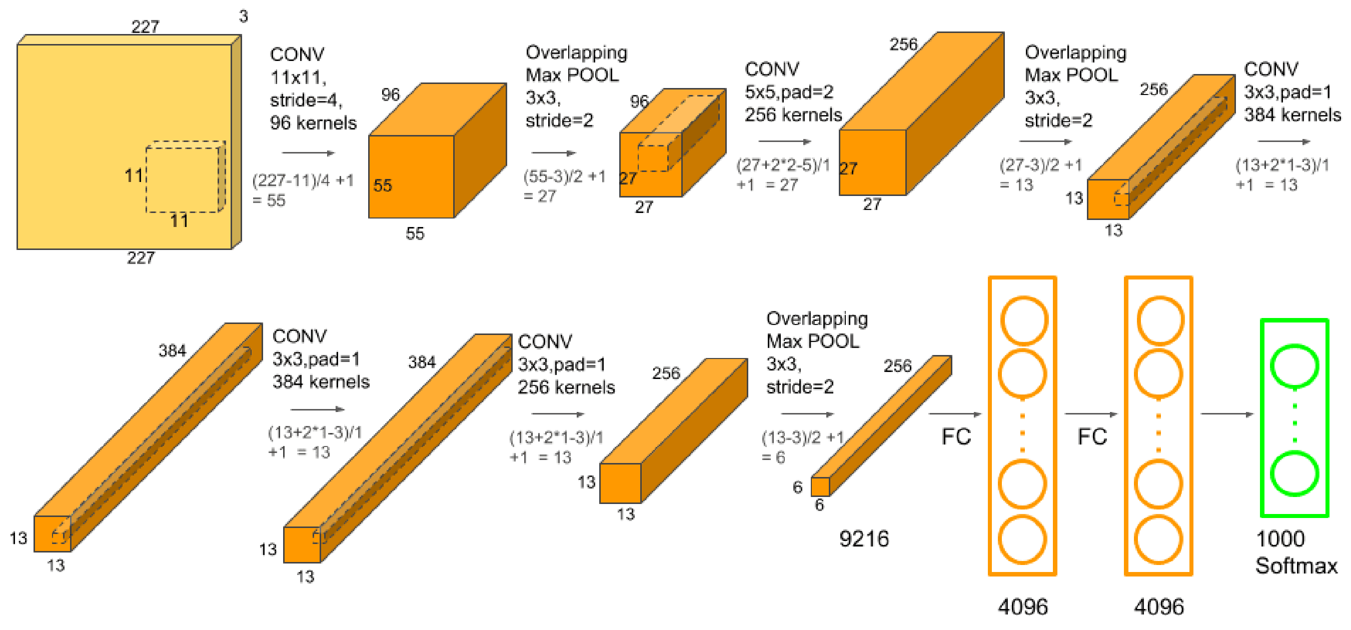
227
3
11
11
227
CONV 11x11, stride=4, 96 kernels
(227-11)/4 +1 = 55
96
55
55
Overlapping Max POOL 3x3, stride=2
(55-3)/2 +1 = 27
96
27
27
CONV 5x5,pad=2 256 kernels
(27+2*2-5)/1 +1 = 27
256
27
27
Overlapping Max POOL 3x3, stride=2
(27-3)/2 +1 = 13
256
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 256 kernels
(13+2*1-3)/1 +1 = 13
256
13
13
Overlapping Max POOL 3x3, stride=2
(13-3)/2 +1 = 6
256
6
6
9216
FC
4096
FC
4096
1000 Softmax

Figure 6 : AlexNet architecture

VGGNet

VGGNet is a CNN architecture developed by the Visual Graphics Group (VGG) at the University of Oxford. VGGNet won the ImageNet Large Scale Visual Recognition Challenge in 2014.

VGGNet consists of the following layers:

1. Convolutional Layer
2. Convolutional Layer
3. Pooling Layer
4. Convolutional Layer
5. Convolutional Layer
6. Pooling Layer
7. Convolutional Layer
8. Convolutional Layer
9. Convolutional Layer
10. Pooling Layer
11. Convolutional Layer
12. Convolutional Layer
13. Convolutional Layer
14. Pooling Layer
15. Fully Connected Layer
16. Fully Connected Layer
17. Output Layer With 1000 Units (Number of Classes)

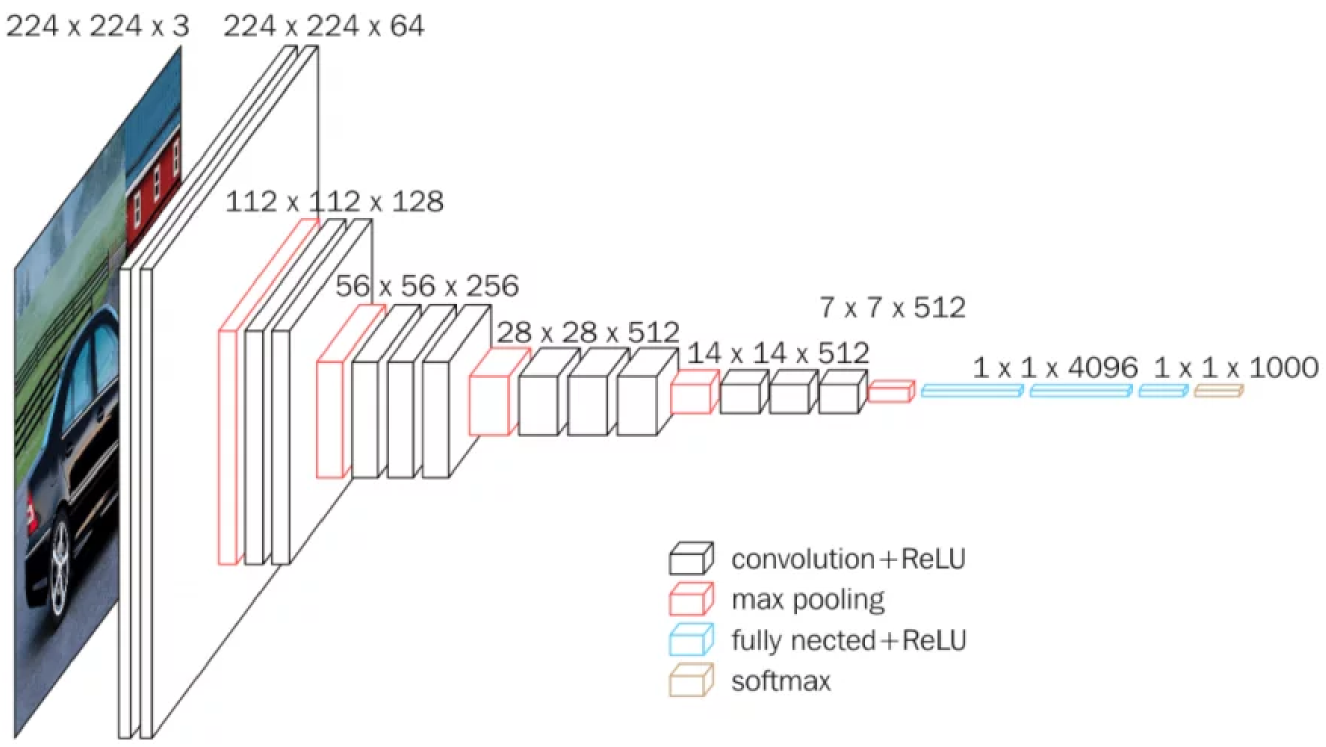

Figure 7 : VGGNet architecture

This might be a good time to mention that the VGGNet architecture is very deep and has a lot of parameters. This makes it computationally expensive to train. but however, it has a very good performance.

GoogLeNet (Inception)

GoogLeNet is a CNN architecture developed by Google. GoogLeNet won the ImageNet Large Scale Visual Recognition Challenge in 2014.

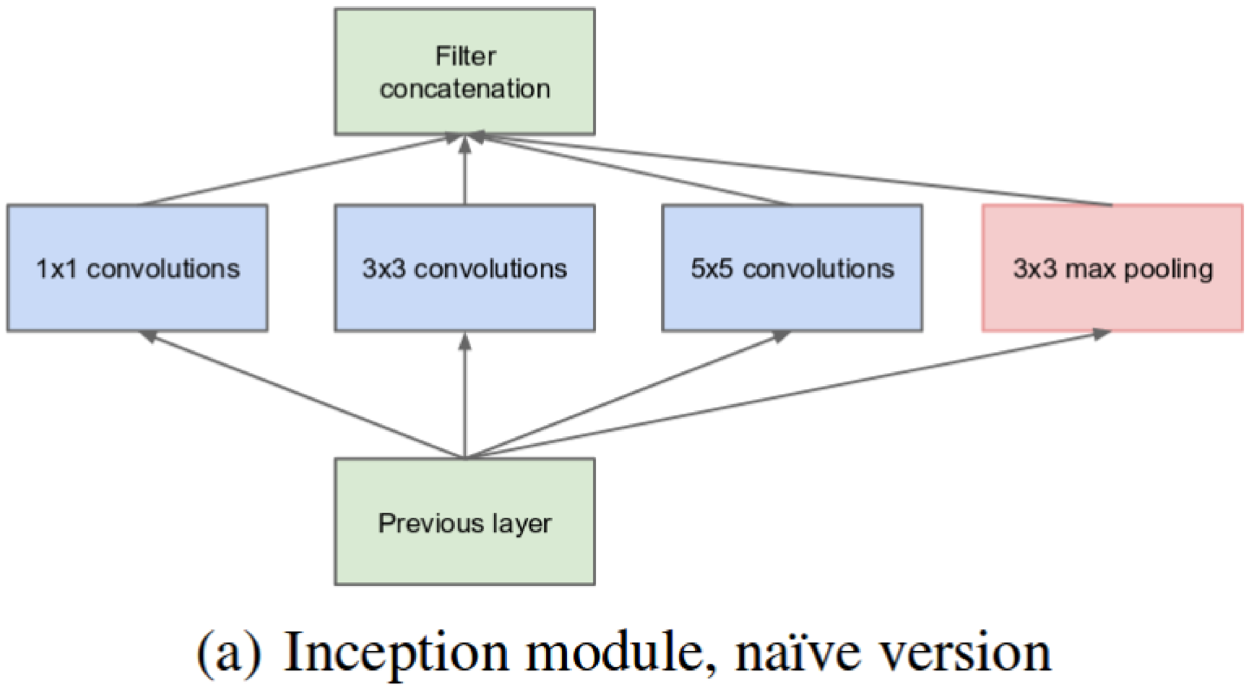

Figure 8 : Inception Module

GoogLeNet consists of the following layers:

1.Many Inception Modules 2.Pooling Layers 3.Fully Connected Layers (in 3 places)

The Inception Module is a combination of different convolutions (1x1, 3x3, 5x5) and pooling operations. The Inception Module is used to extract features from the input image.

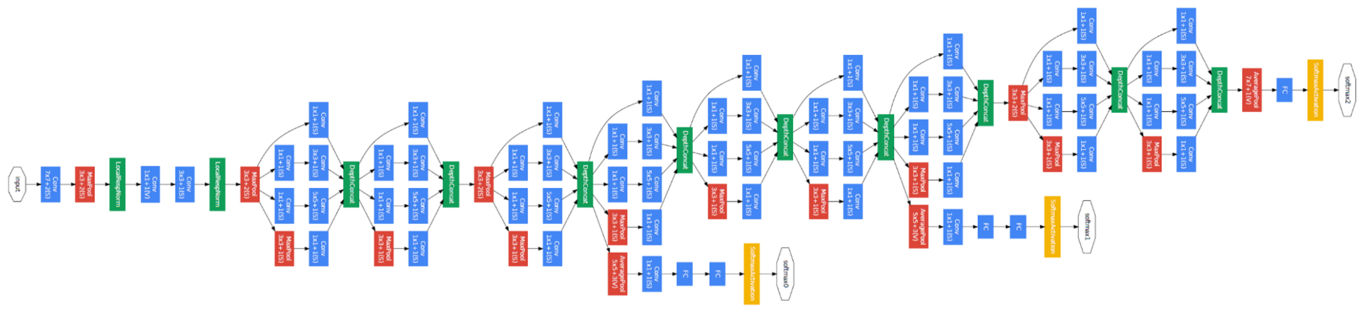

Figure 8 : GoogLeNet architecture

There's various versions of GoogLeNet like InceptionV1, InceptionV2, InceptionV3, InceptionV4, and Inception-ResNet.

ResNet (Residual Networks)

ResNet is a CNN architecture developed by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. ResNet won the ImageNet Large Scale Visual Recognition Challenge in 2015.

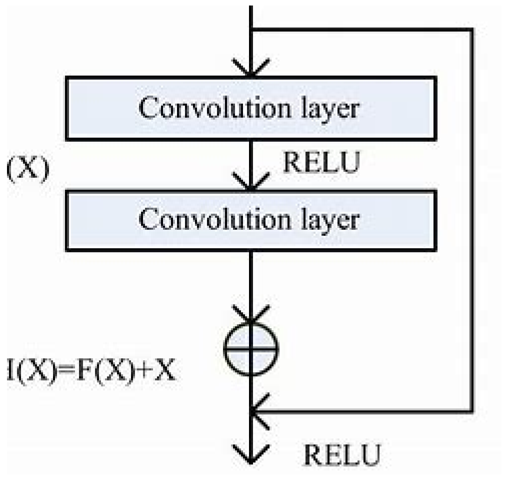

Figure 9 : Residual Block

A Residual block is a block that contains a skip connection. The skip connection skips one or more layers and is added to the output of the layer. The skip connection is used to prevent the vanishing gradient problem.

ResNet consists of the following layers:

1. Convolutional Layer
2. Residual Block
3. Residual Block
4. Residual Block
5. Residual Block
6. Pooling Layer

7. Fully Connected Layer
8. Output Layer With 1000 Units (Number of Classes)

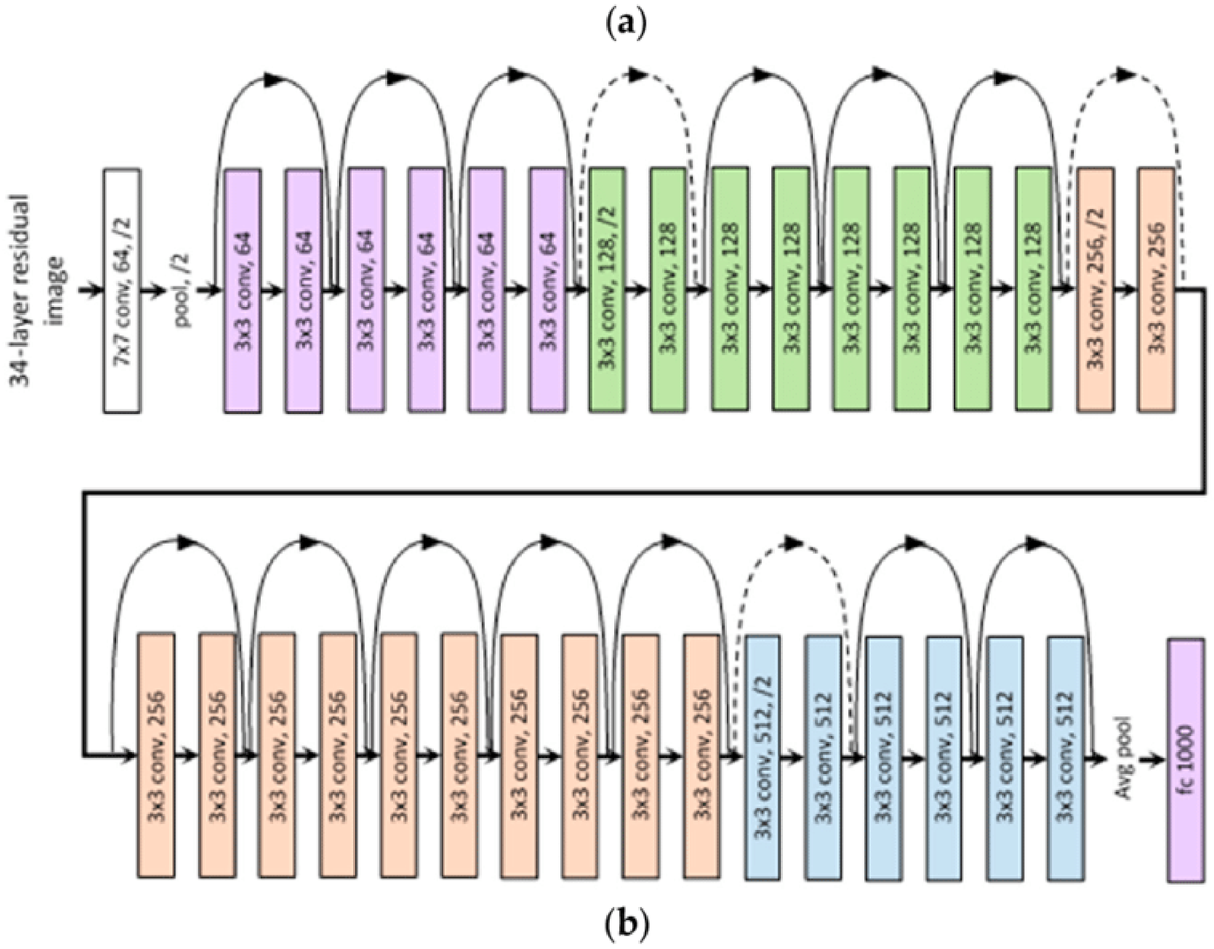

Figure 9 : ResNet architecture

4. Advanced Regularization Techniques

Batch Normalization

Batch normalization is a technique used to normalize the input of each layer. Batch normalization is used to speed up the training process and to reduce overfitting.

it works like the following:

1. Calculate the mean and variance of the input.
2. Normalize the input using the mean and variance.
3. Scale and shift the normalized input using gamma and beta.
4. Update gamma and beta during the training process.
5. Use the normalized input as the input to the activation function.

```
**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
           Parameters to be learned: $\gamma, \beta$
**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

    $\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$    // mini-batch mean
    $\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$    // mini-batch variance
    $\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$    // normalize
    $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$    // scale and shift
```

Figure 10 : Batch normalization

Data Augmentation

Data Augmentation is a technique used to increase the size of the training dataset. Data Augmentation is used to reduce overfitting and to improve the performance of the model.

Data Augmentation involves the following operations:

1. Flipping the image horizontally or vertically.
2. Rotating the image.
3. Zooming the image.
4. Changing the brightness of the image.
5. Changing the contrast of the image.
6. Adding noise to the image.
7. Cropping the image.
8. Resizing the image.

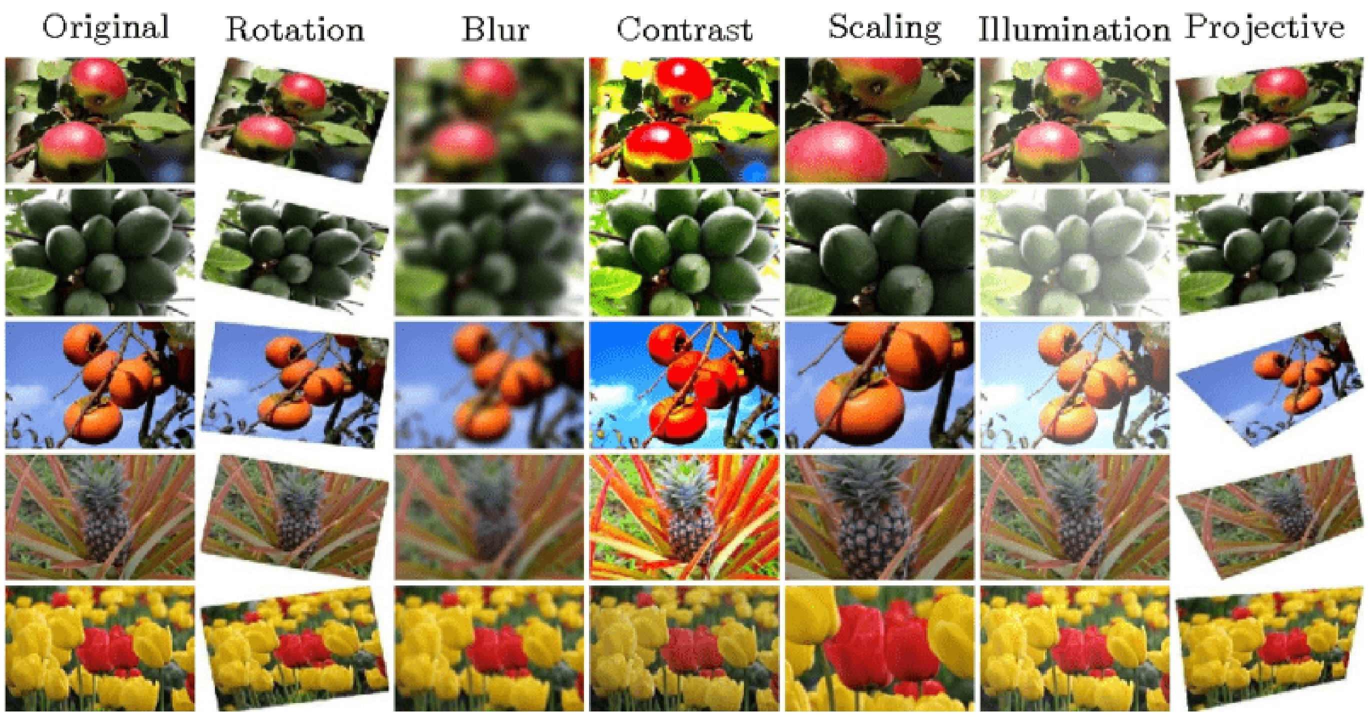

Figure 11 : Data Augmentation

this technique help not picking up on unadequate features and help the model generalize better.