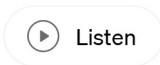


Residual Networks With Examples



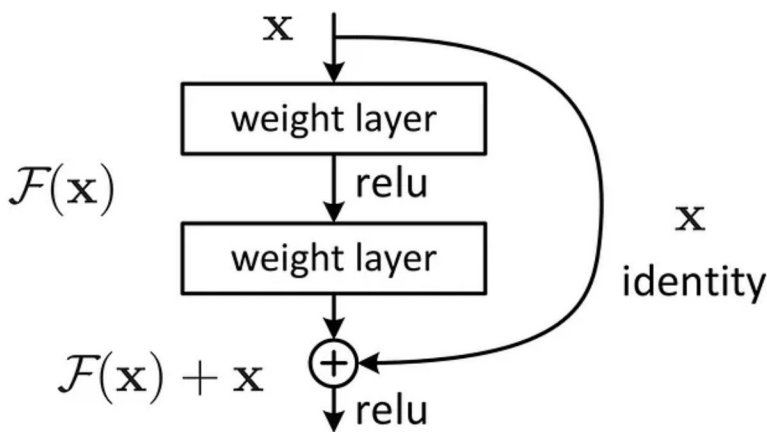
Anas BRITAL · Follow

3 min read · Oct 28, 2021



Residual Networks Explained and Implemented Using Keras and PyTorch .

Residual Networks



In this article we will see what is a Residual network, and we will see two examples of this networks (ResNet 50 and ResNeXt 50), and how to implement them in both keras and PyTorch.

Residual Networks :

The first problem we encounter when we want to train deep neural networks is the gradient vanishing problem, the Residual networks (ResNet) come to solve this problem, ResNet networks are characterized

by skip connections, or shortcuts to jump over some layers, this trick gives the ability to train really deep networks without caring about The problem of gradient vanishing.

Feed Forward propagation in Residual Networks :

$$a^l := \mathbf{g} \left(z^l + \sum_{k=2}^K W^{\ell-k, \ell} \cdot a^{\ell-k} \right)$$

a^l : The Activation of layer L

$W^{\ell-k, \ell}$: The Weights Matrix between layer l-k and l.

g : is The Activation Function of Layer l

And z_l defined like this :

$$z^l = W^{\ell-1, \ell} \cdot a^{\ell-1} + b^l$$

feed Forward in ResNet

Backward propagation in Residual Networks :

$$\Delta w^{\ell-k,\ell} := -\eta \frac{\partial E^{\ell}}{\partial w^{\ell-k,\ell}} = -\eta a^{\ell-k} \cdot \delta^{\ell}$$

- a^{ℓ} : The Activation of layer L
- $w^{\ell-k,\ell}$: The Weights Matrix between layer l-k and l.
- η : is The Learning rate .
- δ^{ℓ} : The Error in Layer l .

backward propagation in ResNet

Examples :

in this section we will see two examples of ResNet Networks , ResNet50 and ResNeXt50 , and we will see how to implement them in keras and PyTorch .

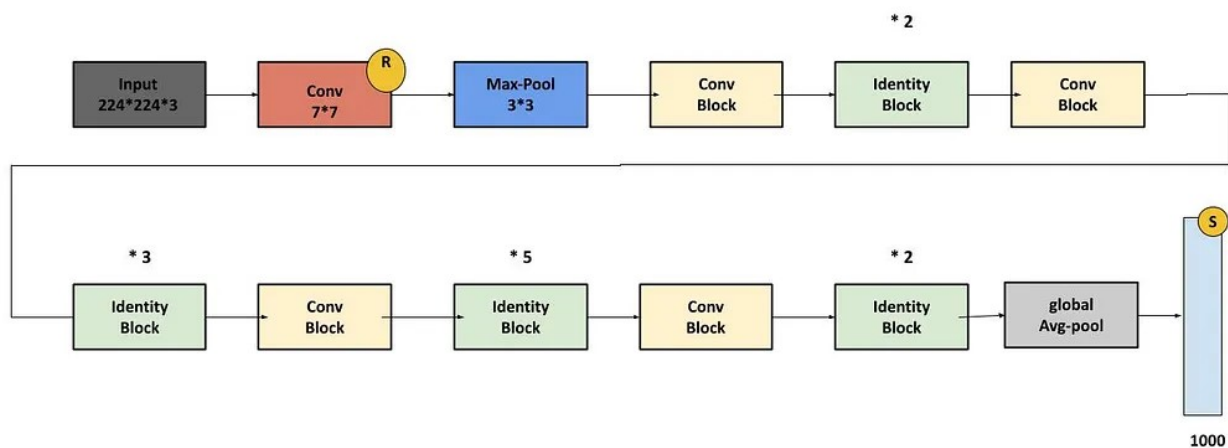
ResNet-50 :

Paper : [Deep Residual Learning for Image Recognition.](#)

Authors : Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Microsoft .

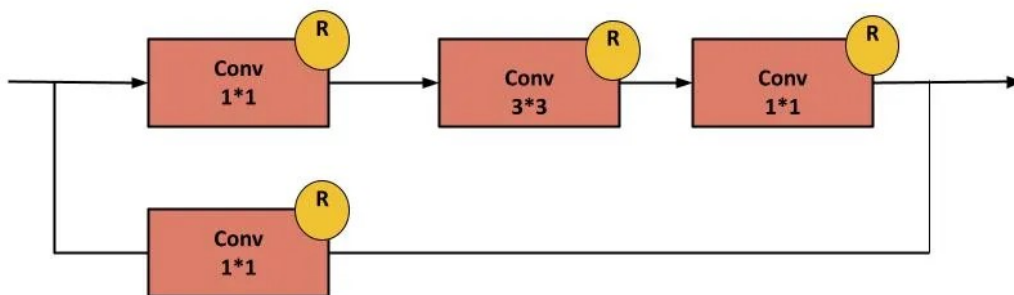
Published in : 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

The Main Architecture :



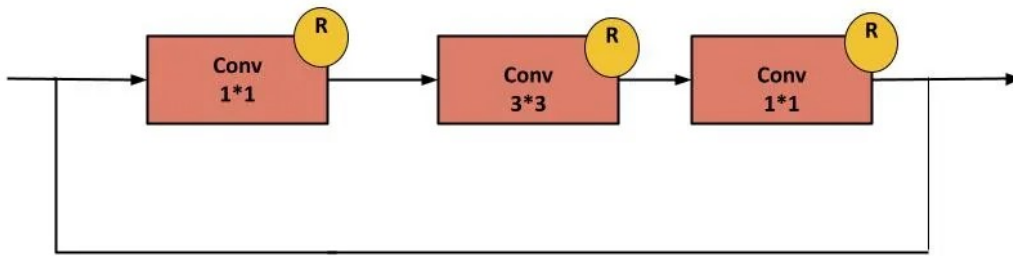
ResNet 50

Conv Block :



Conv Block

Identity Block :



identity Block

Implementation :

1. keras :

```

1  from keras.models import Model
2  from keras.layers.merge import concatenate
3  from keras.layers import Conv2D , MaxPool2D , Input ,AveragePooling2D, Dense , Dropout ,Activation
4
5
6
7  def IdentityBlock(prev_Layer , filters):
8      f1 , f2 , f3 = filters
9
10     x = Conv2D(filters=f1, kernel_size = (1,1) , strides=(1,1), padding='valid')(prev_Layer)
11     x = BatchNormalization(axis=3)(x)
12     x = Activation(activation='relu')(x)
13
14     x = Conv2D(filters=f2, kernel_size = (3,3) , strides=(1,1), padding='same')(x)
15     x = BatchNormalization(axis=3)(x)
16     x = Activation(activation='relu')(x)
17
18     x = Conv2D(filters=f3, kernel_size = (1,1) , strides=(1,1), padding='valid')(x)
19     x = BatchNormalization(axis=3)(x)
20     x = Activation(activation='relu')(x)
21
22     x = concatenate([ x, prev_Layer ], axis=-1)
23     x = Activation(activation='relu')(x)
24     return x
25
26  def ConvBlock(prev_Layer , filters , strides):
27     f1 , f2 , f3 = filters
28
29     #Path 1
30     x = Conv2D(filters=f1, kernel_size = (1,1) ,padding='valid', strides=strides)(prev_Layer)
31     x = BatchNormalization(axis=3)(x)
32     x = Activation(activation='relu')(x)
33
34     x = Conv2D(filters=f2, kernel_size = (3,3) , padding='same' , strides=(1 ,1))(x)
35     x = BatchNormalization(axis=3)(x)
36     x = Activation(activation='relu')(x)
37
38     x = Conv2D(filters=f3, kernel_size = (1,1), padding='valid' , strides=(1 ,1))(x)
39     x = BatchNormalization(axis=3)(x)
40     x = Activation(activation='relu')(x)
41
42     #Path 2
43
44     x2 = Conv2D(filters=f3, kernel_size=(1,1), padding='valid' , strides=strides)(prev_Layer)
45     x2 = BatchNormalization(axis=3)(x2)
46
47     x = concatenate([x , x2], axis=-1)
48     x = Activation(activation='relu')(x)
49     return x
50
51  def ResNet50():
52     input_layer = Input(shape = (224, 224, 3))
53     #----- 1

```

```
53 #Stage 1
54 x = ZeroPadding2D((3, 3))(input_layer)
55 x = Conv2D(filters = 64, kernel_size = (7,7), strides=(2,2)) (x)
56 x = BatchNormalization(axis=3)(x)
57 x = Activation(activation='relu')(x)
58 x = MaxPool2D(pool_size=(3,3) , strides=(2,2))(x)
59
60 #Stage 2
61 x = ConvBlock(prev_Layer=x, filters = [64 , 64 , 256], strides = 1)
62 x = IdentityBlock(prev_Layer=x, filters = [64,64,256])
63 x = IdentityBlock(prev_Layer=x, filters = [64,64,256])
64
65 #Stage 3
66 x = ConvBlock(prev_Layer=x, filters = [128 , 128 , 512], strides = 2)
67 x = IdentityBlock(prev_Layer=x, filters = [128 , 128 , 512])
68 x = IdentityBlock(prev_Layer=x, filters = [128 , 128 , 512])
69 x = IdentityBlock(prev_Layer=x, filters = [128 , 128 , 512])
70
71 #Stage 4
72 x = ConvBlock(prev_Layer=x, filters = [256 , 256 , 1024], strides = 2)
73 x = IdentityBlock(prev_Layer=x, filters = [256 , 265 , 1024])
74 x = IdentityBlock(prev_Layer=x, filters = [256 , 265 , 1024])
75 x = IdentityBlock(prev_Layer=x, filters = [256 , 265 , 1024])
76 x = IdentityBlock(prev_Layer=x, filters = [256 , 265 , 1024])
77 x = IdentityBlock(prev_Layer=x, filters = [256 , 265 , 1024])
78
79 #Stage 5
80 x = ConvBlock(prev_Layer=x, filters = [512 , 512 , 2048], strides = 2)
81 x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 2048])
82 x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 2048])
83
84 #Stage 6
85 x = AveragePooling2D(pool_size=(7,7)) (x)
86
87 x = Flatten()(x)
88 x = Dense(units=1000, activation='softmax')(x)
89
90 model = Model(inputs=input_layer , outputs = x , name='ResNet50')
91 return model
```

2. PyTorch :

```
1 import torch
2 import torch.nn as nn
3
4 class convolution2D(nn.Module):
5     def __init__(self , in_channels , out_channels , kernel_size , stride , padding , activation):
6         super(convolution2D , self).__init__()
7         self.conv = nn.Conv2d(in_channels , out_channels , kernel_size , stride , padding)
8         self.batchNormalization = nn.BatchNorm2d(num_features=out_channels)
9         self.activation = nn.ReLU()
10        self.act = activation
11
12    def forward(self , x):
13        out = self.conv(x)
14        out = self.batchNormalization(out)
15        if self.act :
16            out = self.activation(out)
17        return out
18
19 class conv_Block(nn.Module):
20     def __init__(self , in_channels , filters , stride):
21         super(conv_Block , self).__init__()
22         f1 , f2 , f3 = filters
23
24         self.branch1 = nn.Sequential(
25             convolution2D(in_channels , f1 , 1 , stride , 0 , True),
26             convolution2D(f1 , f2 , 3 , 1 , 1 , True),
27             convolution2D(f2 , f3 , 1 , 1 , 0 , False)
28         )
29
30         self.branch2 = convolution2D(in_channels , f3 , 1 , stride , 0 , False)
31
32         self.activation = nn.ReLU()
33
34     def forward(self , x):
35
36         branch1 = self.branch1(x)
37
38         branch2 = self.branch2(x)
39
40         out = torch.cat([branch1 , branch2] , 1)
41
42         return self.activation(out)
43
44 class identity_Block(nn.Module):
45     def __init__(self , in_channels , filters ):
46         super(identity_Block , self).__init__()
47
48         f1 , f2 , f3 = filters
49         self.branch1 = nn.Sequential(
50             convolution2D(in_channels , f1 , 1 , 1 , 0 , True),
51             convolution2D(f1 , f2 , 3 , 1 , 1 , True),
52             convolution2D(f2 , f3 , 1 , 1 , 0 , False)
53         )
54         self.act = nn.ReLU()
```

```
53     )
54
55     self.activation = nn.ReLU()
56
57     def forward(self , x):
58
59         branch1 = self.branch1(x)
60
61         branch2 = x
62
63         out = torch.cat([branch1 , branch2] , 1)
64
65         return self.activation(out)
66
67     class ResNet_50(nn.Module):
68         def __init__(self):
69             super(ResNet_50 , self).__init__()
70
71             self.conv1 = convolution2D(in_channels = 3 , out_channels = 64, kernel_size = 7 , stride =
72             self.MaxPooling1 = nn.MaxPool2d(kernel_size=3 , stride = 2)
73
74             self.ConvBlock1 = conv_Block(64 , [64, 64,256] , 1)
75             self.IdentityBlock1 = identity_Block(512 , [64,64,256])
76             self.IdentityBlock2 = identity_Block(768 , [64,64,256])
77             self.ConvBlock2 = conv_Block(1024 , [128, 128,512] , 2)
78             self.IdentityBlock3 = identity_Block(1024 , [128,128,512])
79             self.IdentityBlock4 = identity_Block(1536 , [128,128,512])
80             self.IdentityBlock5 = identity_Block(2048 , [128,128,512])
81             self.ConvBlock3 = conv_Block(2560 , [256, 256,1024] , 2)
82             self.IdentityBlock6 = identity_Block(2048 , [256, 256,1024])
83             self.IdentityBlock7 = identity_Block(3072 , [256, 256,1024])
84             self.IdentityBlock8 = identity_Block(4096 , [256, 256,1024])
85             self.IdentityBlock9 = identity_Block(5120 , [256, 256,1024])
86             self.IdentityBlock10 = identity_Block(6144 , [256, 256,1024])
87             self.ConvBlock4 = conv_Block(7168 , [512, 512,2048] , 2)
88             self.IdentityBlock11 = identity_Block(4096 , [512, 512,2048])
89             self.IdentityBlock12 = identity_Block(6144 , [512, 512,2048])
90             self.globalAvgPooling = nn.Conv2d(in_channels=8192 , out_channels=8192 , kernel_size=7)
91             self.fc = nn.Linear(in_features=8192 , out_features=1000)
92             self.activation = nn.Softmax()
93
94         def forward(self , x):
95
96             out = self.conv1(x)
97             out = self.MaxPooling1(out)
98
99             out = self.ConvBlock1(out)
100
101             out = self.IdentityBlock1(out)
102             out = self.IdentityBlock2(out)
103
104             out = self.ConvBlock2(out)
105
106             out = self.IdentityBlock3(out)
107             out = self.IdentityBlock4(out)
```

```
108     out = self.IdentityBlock5(out)
109
110     out = self.ConvBlock3(out)
111
112     out = self.IdentityBlock6(out)
113     out = self.IdentityBlock7(out)
114     out = self.IdentityBlock8(out)
115     out = self.IdentityBlock9(out)
116     out = self.IdentityBlock10(out)
117
118     out = self.ConvBlock4(out)
119
120     out = self.IdentityBlock11(out)
121     out = self.IdentityBlock12(out)
122
123     out = self.globalAvgPooling(out)
124
125     out = out.reshape(out.shape[0] , -1)
126
127     out = self.fc(out)
128     out = self.activation(out)
129
130     return out
```

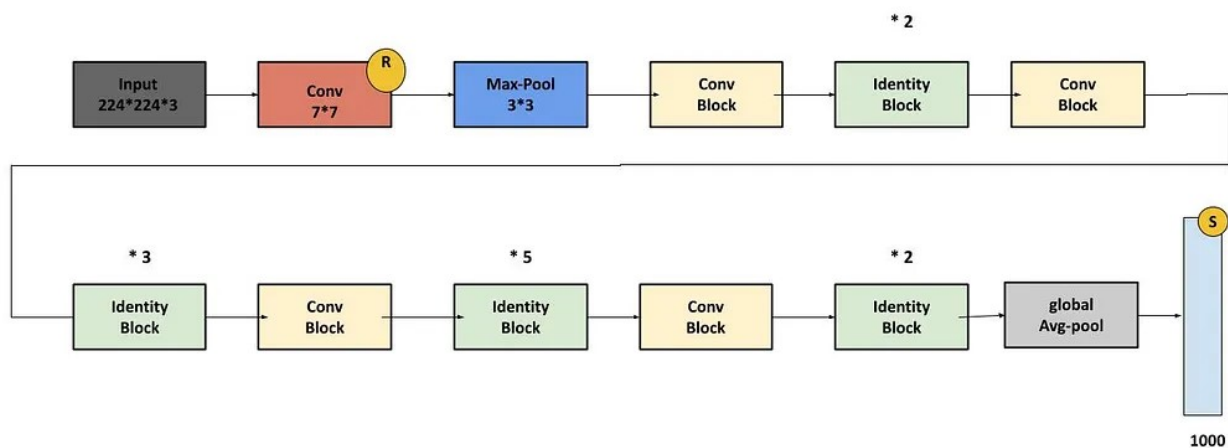
ResNeXt-50 :

Paper : [Aggregated Residual Transformations for Deep Neural Networks.](#)

Authors : Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He. University of California San Diego, Facebook Research .

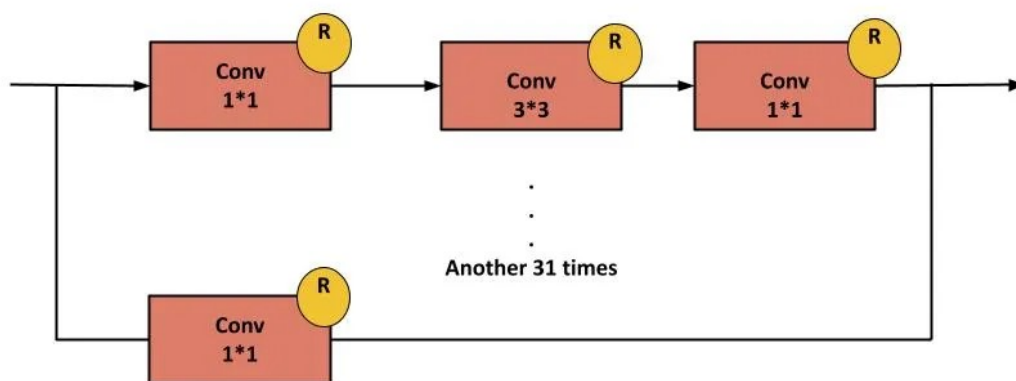
Published in : 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) .

The Main Architecture :



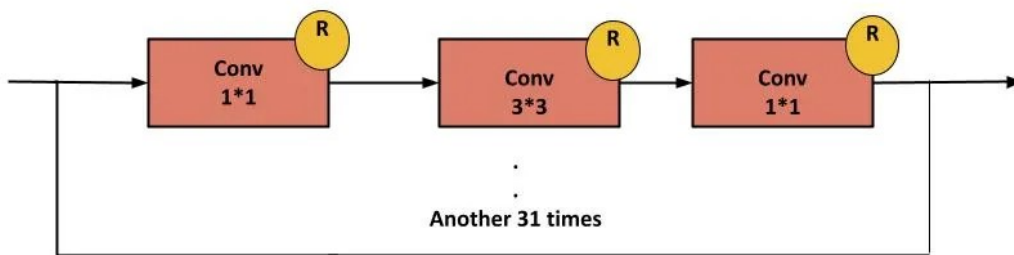
ResNeXt-50

Conv Block :



Conv Block

Identity Block :



identity Block

Implementation :

1. Keras :

```

1  from keras.models import Model
2  from keras.layers import Conv2D , MaxPool2D , ZeroPadding2D, Input ,AveragePooling2D, Dense , D
3  from keras.layers import Add
4
5  def IdentityBlock(prev_Layer , filters):
6
7      f1 , f2 ,f3 = filters
8      block = []
9
10     for i in range(32):
11         x = Conv2D(filters=f1, kernel_size = (1,1) , strides=(1,1), padding='valid')(prev_Layer
12         x = BatchNormalization(axis=3)(x)
13         x = Activation(activation='relu')(x)
14
15         x = Conv2D(filters=f2, kernel_size = (3,3) , strides=(1,1), padding='same')(x)
16         x = BatchNormalization(axis=3)(x)
17         x = Activation(activation='relu')(x)
18
19         x = Conv2D(filters=f3, kernel_size = (1,1) , strides=(1,1), padding='valid')(x)
20         x = BatchNormalization(axis=3)(x)
21         x = Activation(activation='relu')(x)
22         block.append(x)
23
24     block.append(prev_Layer)
25     x = Add()(block)
26     x = Activation(activation='relu')(x)
27
28     return x
29
30
31 def ConvBlock(prev_Layer , filters , strides):
32     f1 , f2 , f3 = filters
33
34     block = []
35
36     for i in range(32):
37         x = Conv2D(filters=f1, kernel_size = (1,1) ,padding='valid' , strides=strides)(prev_Layer
38         x = BatchNormalization(axis=3)(x)
39         x = Activation(activation='relu')(x)
40
41         x = Conv2D(filters=f2, kernel_size = (3,3) , padding='same' , strides=(1 ,1))(x)
42         x = BatchNormalization(axis=3)(x)
43         x = Activation(activation='relu')(x)
44
45         x = Conv2D(filters=f3, kernel_size = (1,1), padding='valid' , strides=(1 ,1))(x)
46         x = BatchNormalization(axis=3)(x)
47         x = Activation(activation='relu')(x)
48         block.append(x)
49
50     x2 = Conv2D(filters=f3, kernel_size=(1,1), padding='valid' , strides=strides)(prev_Layer)
51     x2 = BatchNormalization(axis=3)(x2)
52
53     block.append(x2)

```

```
53     block.append(x2)
54     x = Add()(block)
55     x = Activation(activation='relu')(x)
56     return x
57
58
59 def ResNext():
60     input_layer = Input(shape = (224, 224, 3))
61     #Stage 1
62     x = ZeroPadding2D((3, 3))(input_layer)
63     x = Conv2D(filters = 64, kernel_size = (7,7), strides=(2,2)) (x)
64     x = BatchNormalization(axis=3)(x)
65     x = Activation(activation='relu')(x)
66     x = MaxPool2D(pool_size=(3,3) , strides=(2,2))(x)
67
68     #Stage 2
69     x = ConvBlock(prev_Layer=x, filters = [128 , 128 , 256], strides = 1)
70     x = IdentityBlock(prev_Layer=x, filters = [128 , 128 , 256])
71     x = IdentityBlock(prev_Layer=x, filters = [128 , 128 , 256])
72
73     #Stage 3
74     x = ConvBlock(prev_Layer=x, filters = [256 , 256 , 512], strides = 2)
75     x = IdentityBlock(prev_Layer=x, filters = [256 , 256 , 512])
76     x = IdentityBlock(prev_Layer=x, filters = [256 , 256 , 512])
77     x = IdentityBlock(prev_Layer=x, filters = [256 , 256 , 512])
78
79     #Stage 4
80     x = ConvBlock(prev_Layer=x, filters = [512 , 512 , 1024], strides = 2)
81     x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 1024])
82     x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 1024])
83     x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 1024])
84     x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 1024])
85     x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 1024])
86
87     #Stage 5
88     x = ConvBlock(prev_Layer=x, filters = [1024 , 1024 , 2048], strides = 2)
89     x = IdentityBlock(prev_Layer=x, filters = [1024 , 1024 , 2048])
90     x = IdentityBlock(prev_Layer=x, filters = [1024 , 1024 , 2048])
91
92     #Stage 6
93     x = AveragePooling2D(pool_size=(7,7)) (x)
94
95     x = Flatten()(x)
96     x = Dense(units=1000, activation='softmax')(x)
97
98     model = Model(inputs=input_layer , outputs = x , name='ResNet50')
99     return model
```

2. PyTorch :

```
1 import torch
2 import torch.nn as nn
3 from torchsummary import summary
4
5 class convolution2D(nn.Module):
6     def __init__(self , in_channels , out_channels , kernel_size , stride , padding , activation):
7         super(convolution2D , self).__init__()
8         self.conv = nn.Conv2d(in_channels , out_channels , kernel_size , stride , padding)
9         self.batchNormalization = nn.BatchNorm2d(num_features=out_channels)
10        self.activation = nn.ReLU()
11        self.act = activation
12
13    def forward(self , x):
14        out = self.conv(x)
15        out = self.batchNormalization(out)
16        if self.act :
17            out = self.activation(out)
18        return out
19
20    class Conv_Block(nn.Module):
21
22        def __init__(self , in_channels , filters , stride):
23            super(Conv_Block , self).__init__()
24            f1 , f2 , f3 = filters
25
26            self.branch1 = nn.Sequential(
27                convolution2D(in_channels , f1 , 1 , stride , 0 , True),
28                convolution2D(f1 , f2 , 3 , 1 , 1 , True),
29                convolution2D(f2 , f3 , 1 , 1 , 0 , False)
30            )
31
32            self.branch2 = convolution2D(in_channels , f3 , 1 , 1 , 0 , False)
33            self.activation = nn.ReLU()
34
35        def forward(self,x):
36
37            paths = [self.branch1(x) for i in range(32)]
38            paths.append(self.branch2(x))
39
40            out = torch.cat(paths , 1)
41            return self.activation(out)
42
43    class Identity_Block(nn.Module):
44
45        def __init__(self , in_channels , filters):
46            super(Identity_Block , self).__init__()
47            f1 , f2 , f3 = filters
48
49            self.branch1 = nn.Sequential(
50                convolution2D(in_channels , f1 , 1 , 1 , 0 , True),
51                convolution2D(f1 , f2 , 3 , 1 , 1 , True),
52                convolution2D(f2 , f3 , 1 , 1 , 0 , False)
53            )
54            self.act = nn.ReLU()
```

```
53     )
54     self.activation = nn.ReLU()
55     def forward(self , x):
56         paths = [self.branch1(x) for i in range(32)]
57         paths.append(x)
58
59         out = torch.cat(paths , 1)
60         return self.activation(out)
61
62     class ResNext_50(nn.Module):
63
64     def __init__(self):
65         super(ResNext_50,self).__init__()
66         self.conv1 = convolution2D(in_channels = 3 , out_channels = 64, kernel_size = 7 , stride =
67         self.MaxPooling1 = nn.MaxPool2d(kernel_size=3 , stride = 2)
68
69         self.convBlock1 = Conv_Block(64 , [128 , 128 , 256] , 1)
70         self.IdentityBlock1 = Identity_Block(8448 , [128 , 128 , 256])
71         self.IdentityBlock2 = Identity_Block(16640 , [128 , 128 , 256])
72         self.convBlock2 = Conv_Block(16896 , [256 , 256 , 512] , 2)
73         self.IdentityBlock3 = Identity_Block(33280 , [256 , 256 , 512])
74         self.IdentityBlock4 = Identity_Block(49664 , [256 , 256 , 512])
75         self.IdentityBlock5 = Identity_Block(66048 , [256 , 256 , 512])
76         self.convBlock3 = Conv_Block(33792 , [512 , 512 , 1024] , 2)
77         self.IdentityBlock6 = Identity_Block(66560 , [512 , 512 , 1024])
78         self.IdentityBlock7 = Identity_Block(99328 , [512 , 512 , 1024])
79         self.IdentityBlock8 = Identity_Block(132096 , [512 , 512 , 1024])
80         self.IdentityBlock9 = Identity_Block(164864 , [512 , 512 , 1024])
81         self.IdentityBlock10 = Identity_Block(197632 , [512 , 512 , 1024])
82         self.convBlock4 = Conv_Block(67584 , [1024 , 1024 , 2048] , 2)
83         self.IdentityBlock11 = Identity_Block(133120 , [1024 , 1024 , 2048])
84         self.IdentityBlock12 = Identity_Block(192656 , [1024 , 1024 , 2048])
85         self.globalAvgPooling = nn.Conv2d(in_channels=8192 , out_channels=8192 , kernel_size=7)
86         self.fc = nn.Linear(in_features=192656 , out_features=1000)
87         self.activation = nn.Softmax()
88     def forward(self , x):
89         out = self.conv1(x)
90         out = self.MaxPooling1(out)
91
92         out = self.convBlock1(out)
93
94         out = self.IdentityBlock1(out)
95         out = self.IdentityBlock2(out)
96
97         out = self.convBlock2(out)
98
99         out = self.IdentityBlock3(out)
100        out = self.IdentityBlock4(out)
101        out = self.IdentityBlock5(out)
102
103        out = self.convBlock3(out)
104
105        out = self.IdentityBlock6(out)
106        out = self.IdentityBlock7(out)
107        out = self.IdentityBlock8(out)
```

```
108     out = self.IdentityBlock9(out)
109     out = self.IdentityBlock10(out)
110
111     out = self.convBlock4(out)
112
113     out = self.IdentityBlock11(out)
114     out = self.IdentityBlock12(out)
115
116     out = self.globalAvgPooling(out)
117
118     out = out.reshape(out.shape[0] , -1)
119
120     out = self.fc(out)
121     out = self.activation(out)
122     return out
```

References :

- If you want to see other Architectures implemented in both PyTorch and Keras you can check this [repo](#) and you can also find high quality images (svg format) of the above illustrations.

- [Residual neural network by wikipedia](#) .
- [PyTorch documentation](#) .
- [Keras documentation](#) .

Resnet Cnn Computer Vision Pytorch Keras



Follow

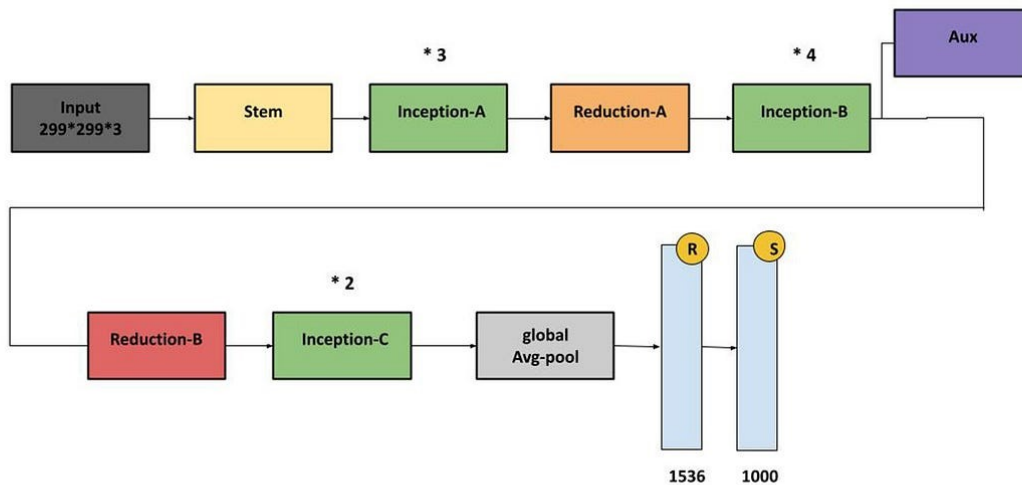
Written by Anas BRITAL


60 Followers

AI and Math Enthusiast (Personal Blog : anasbrital98.github.io/) .

More from Anas BRITAL

Inception V3





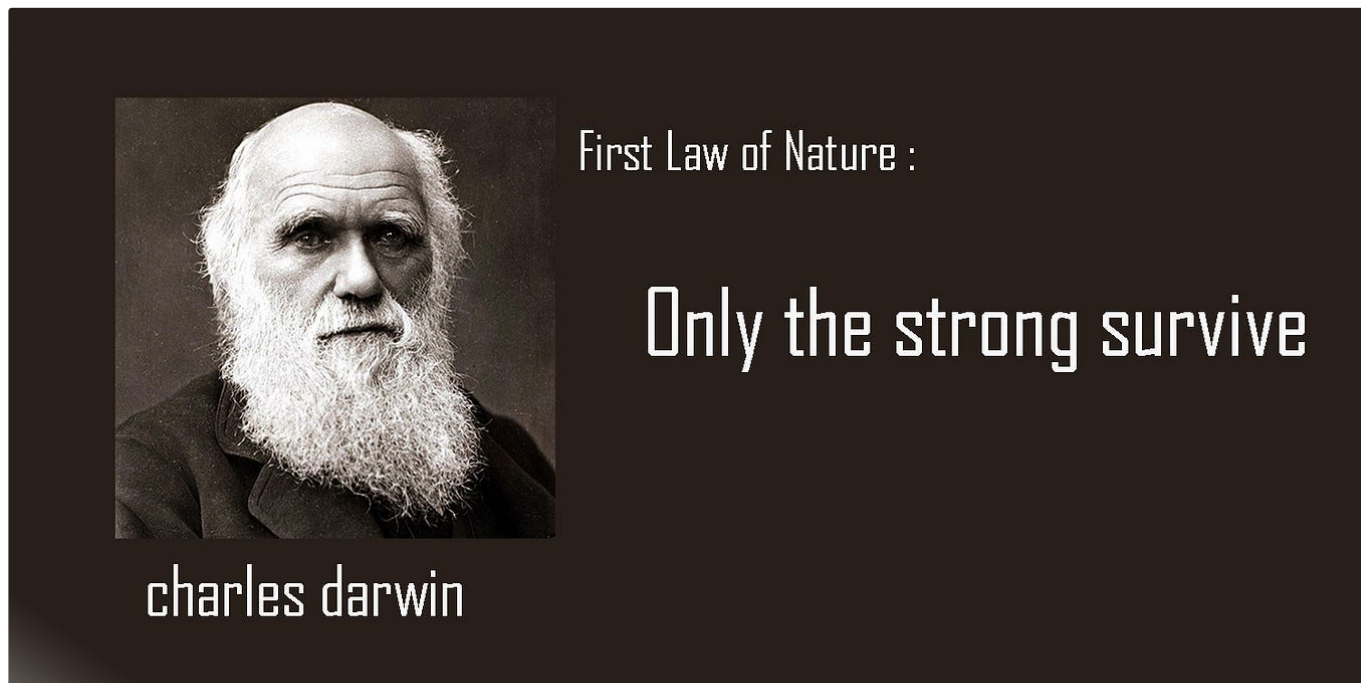
 Anas BRITAL


Inception V3 CNN Architecture Explained .

Inception-V3 CNN Architecture illustrated and Implemented in both Keras and PyTorch .

4 min read · Oct 23, 2021

 12 





 Anas BRITAL

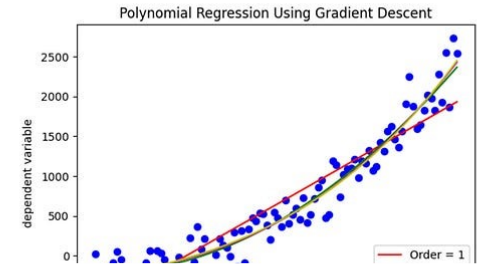
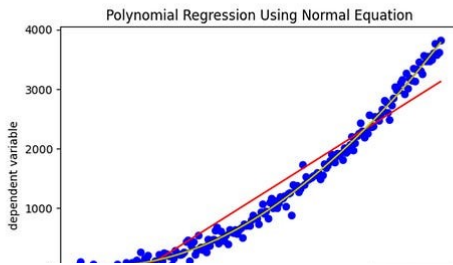
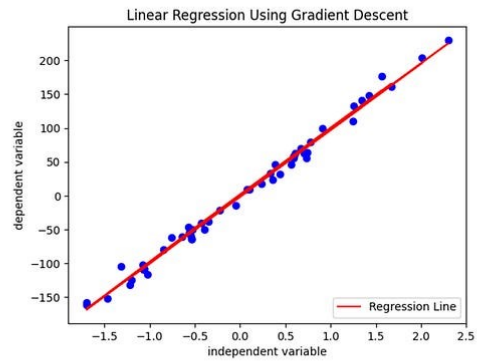
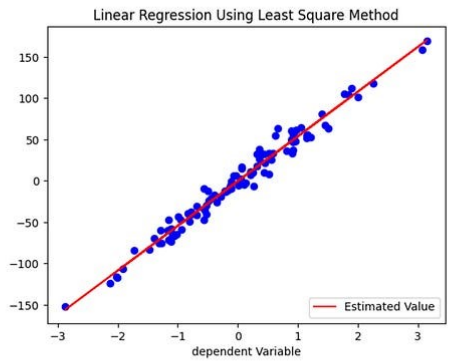
Genetic Algorithm Explained :

Everything you need to know About Genetic Algorithm .

10 min read · Oct 16, 2021

 113  2





Anas BRITAL

Regression Analysis

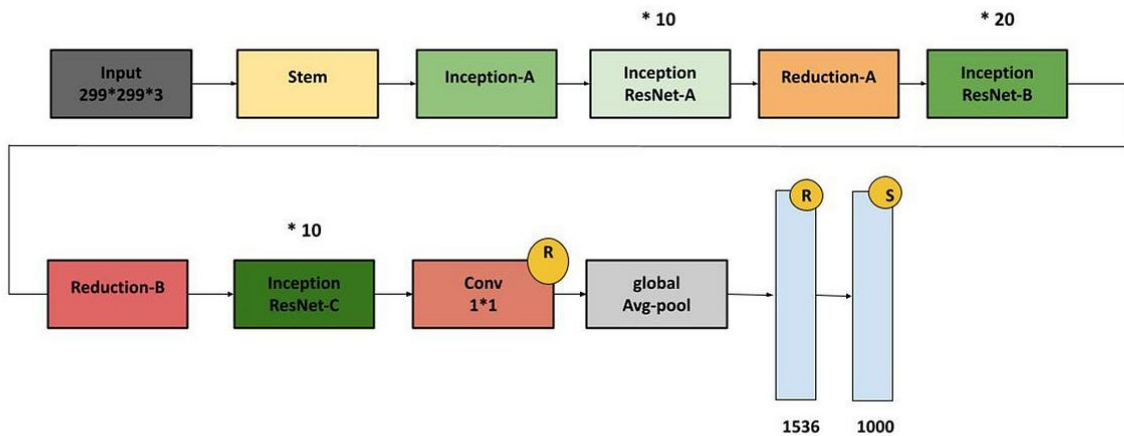
Regression analysis models Explained and Implemented Using Python

5 min read · Nov 6, 2021

55



Inception-V2



Anas BRITAL

Inception V2 CNN Architecture Explained .

Inception-V2 CNN Architecture illustrated and Implemented in both Keras and PyTorch .

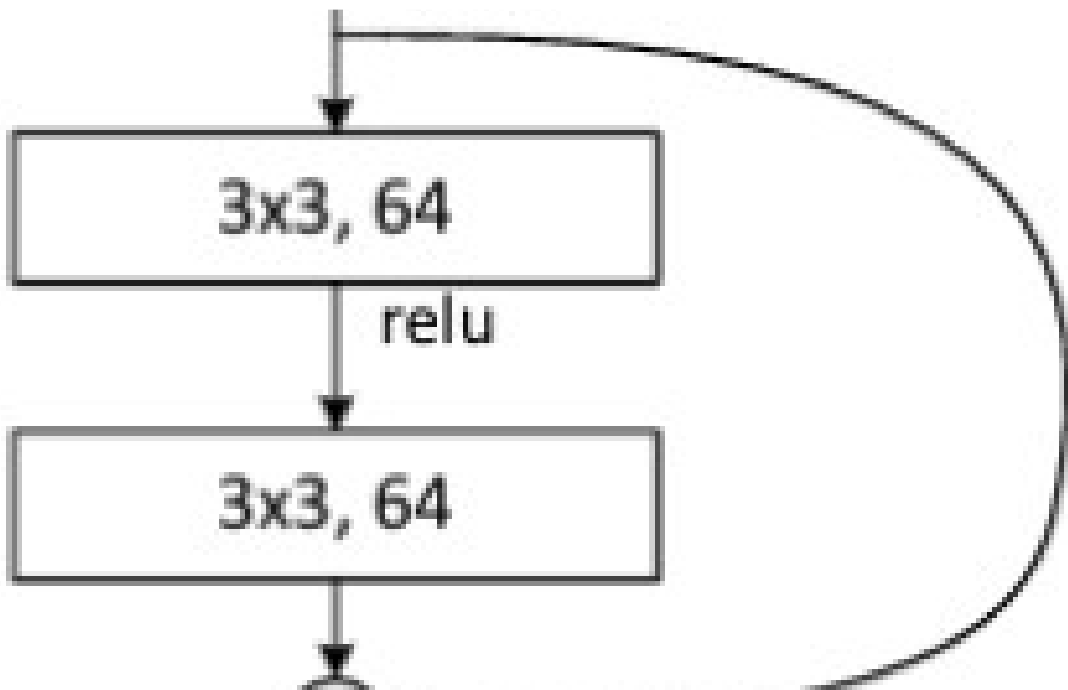
4 min read · Oct 23, 2021

5



See all from Anas BRITAL

Recommended from Medium



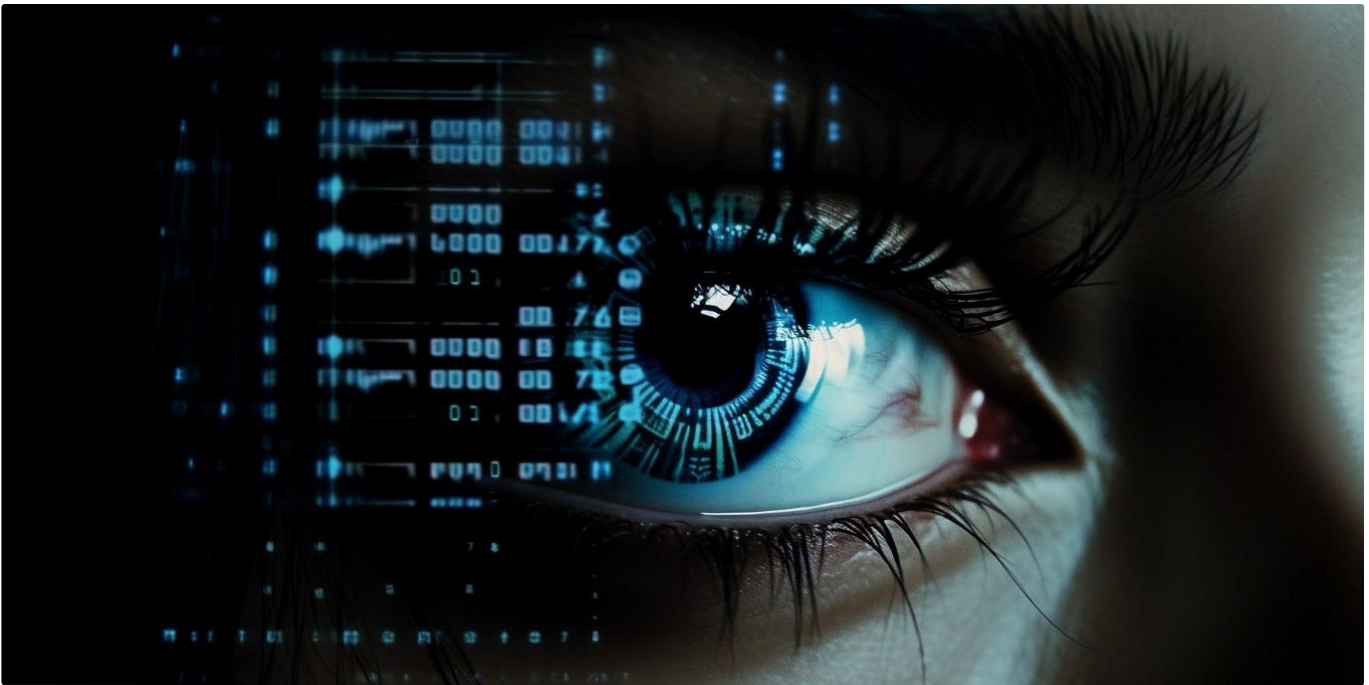
Chen-Yu Chang

Building a Customized Residual CNN with PyTorch

Residual networks (ResNets) have revolutionized the field of deep learning by enabling the construction of much deeper networks than was...

3 min read · Nov 5, 2023







 Azeem - I

Understanding ResNet Architecture: A Deep Dive into Residual Neural Network


Residual Network is a deep Learning model used for computer vision applications. The ResNet (Residual Neural Network) architecture was...

6 min read · Nov 14, 2023

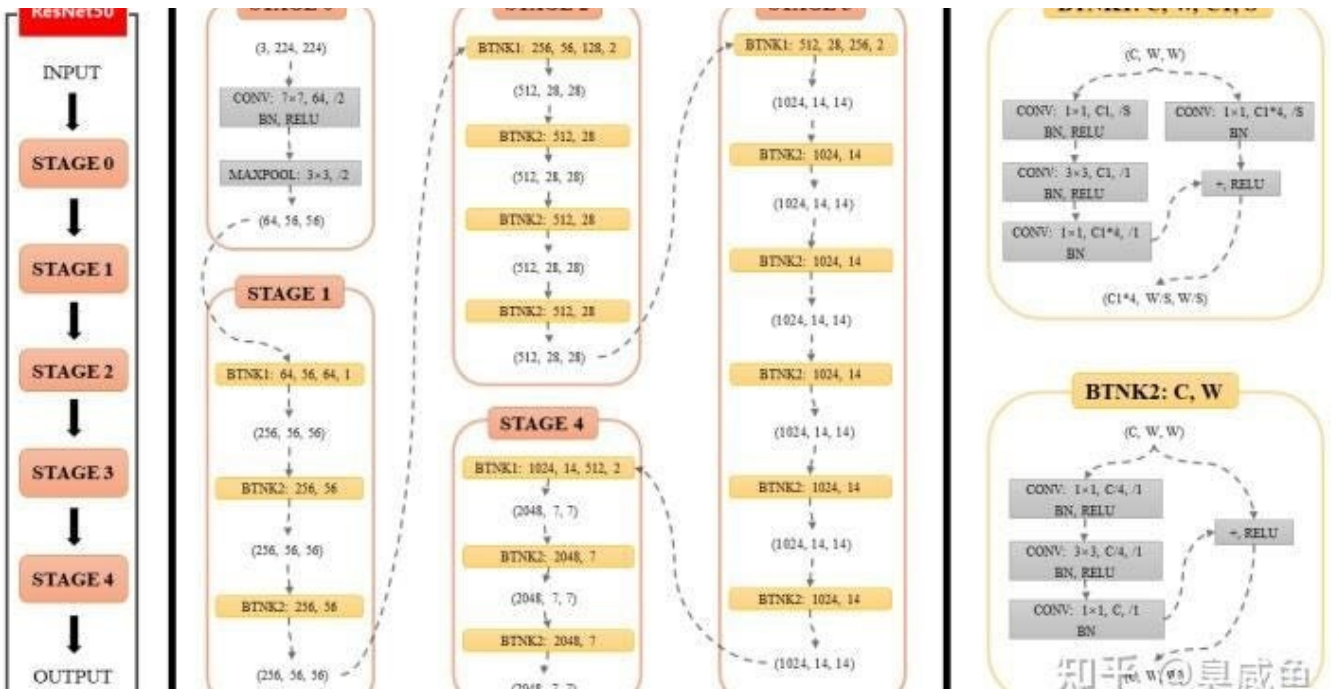
 102 



Lists

 **Practical Guides to Machine Learning**
10 stories · 1312 saves

 **Natural Language Processing**
1369 stories · 865 saves



noplaxochia

Pytorch ResNet from Scratch

Pytorch implementation of

1min read · Jan 15, 2024



What are the query, key, and value vectors?

A Simplified Explanation



[Learn LLM](#)

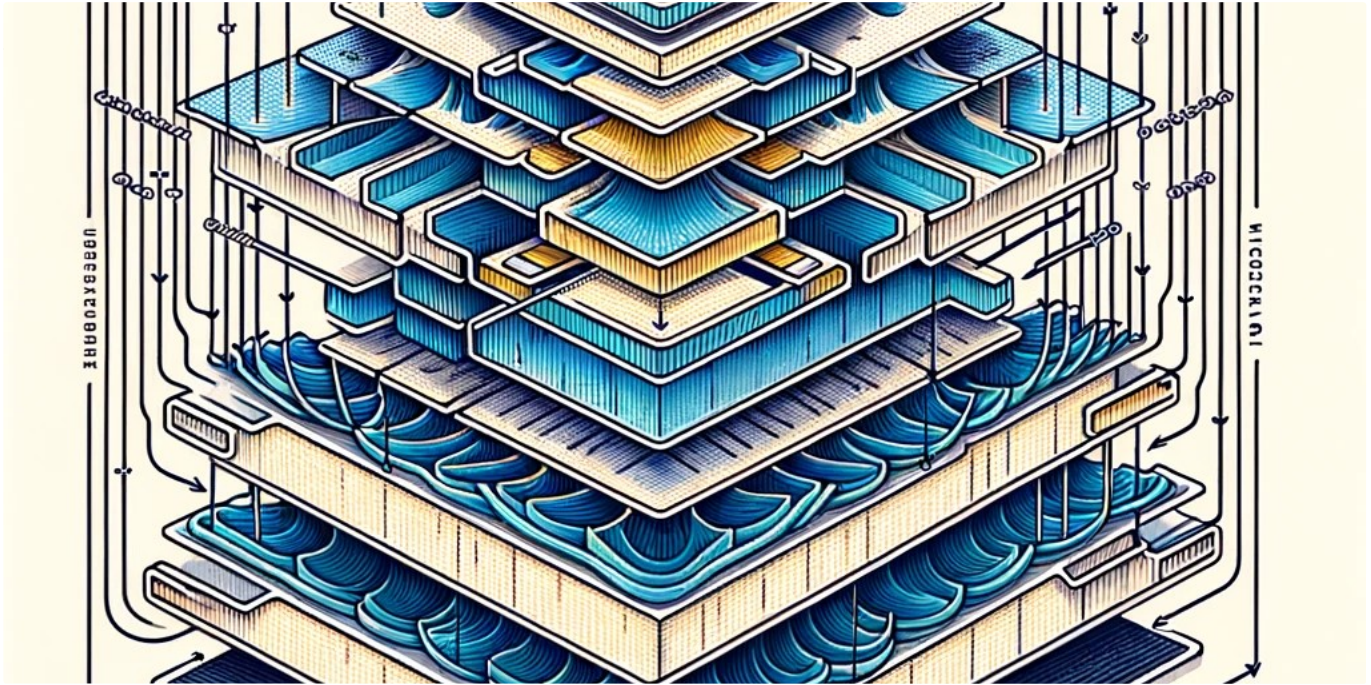
 RAHULRAJ P V


What are the query, key, and value vectors?

In a transformer architecture, “key,” “query,” and “value” are fundamental components used in the mechanism of attention. Attention is an...

5 min read · Dec 10, 2023

 60  1





 Bragadeesh Sundararajan

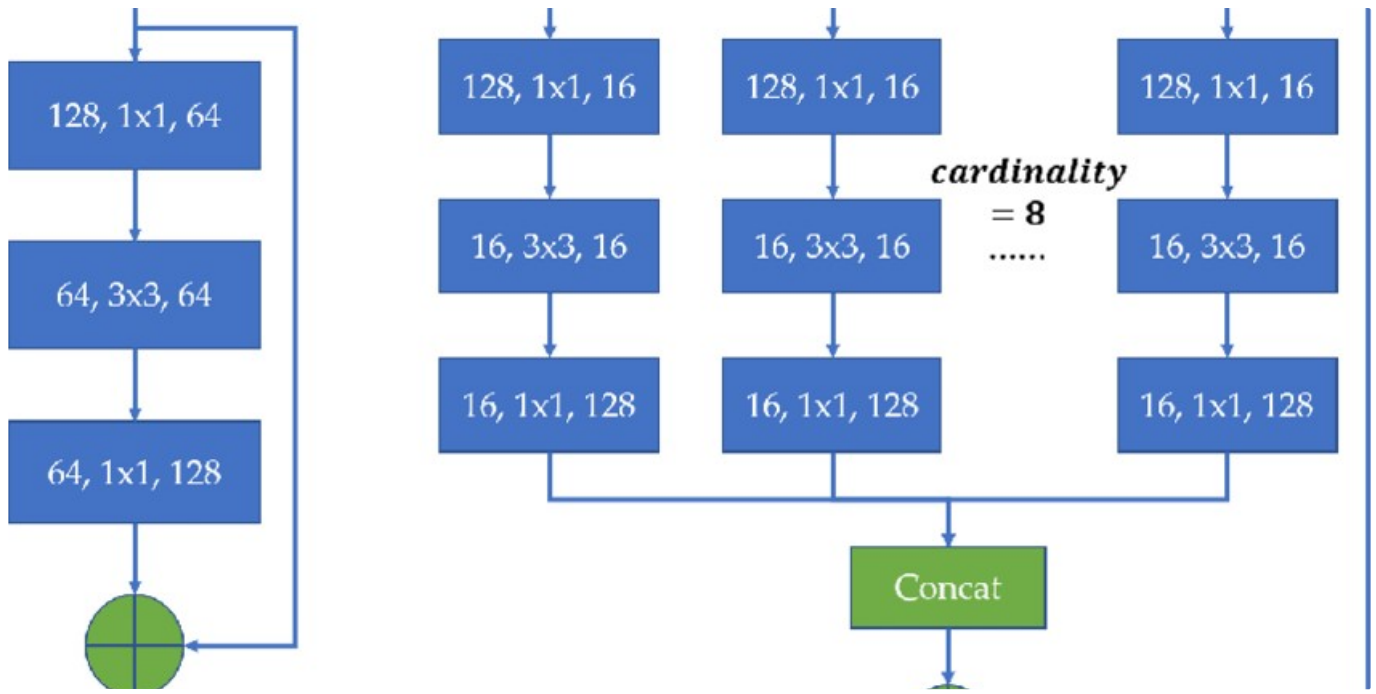
Decoding ResNet: Revolutionizing Deep Learning Architectures

In the dynamic world of machine learning and artificial intelligence, the evolution of deep neural networks has been nothing short of a...

🌟 · 10 min read · Jan 21, 2024

 1 





Atakan Erdoğan

ResNeXt: A New Paradigm in Image Processing

Hello everyone, in this article I'm going to explain about ResNeXt, its deep learning-based structure and usage. In addition, I'm going to...

5 min read · Nov 4, 2023

88



See more recommendations