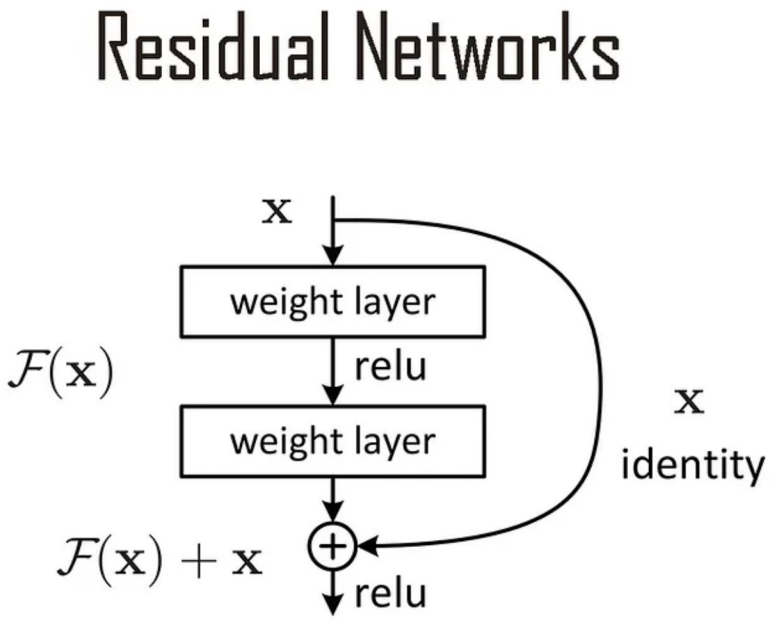
Residual Networks With Examples

Anas BRITAL · Follow
3 min read · Oct 28, 2021

Residual Networks Explained and Implemented Using Keras and PyTorch .



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 :

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

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 :

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

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.

n : 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 .

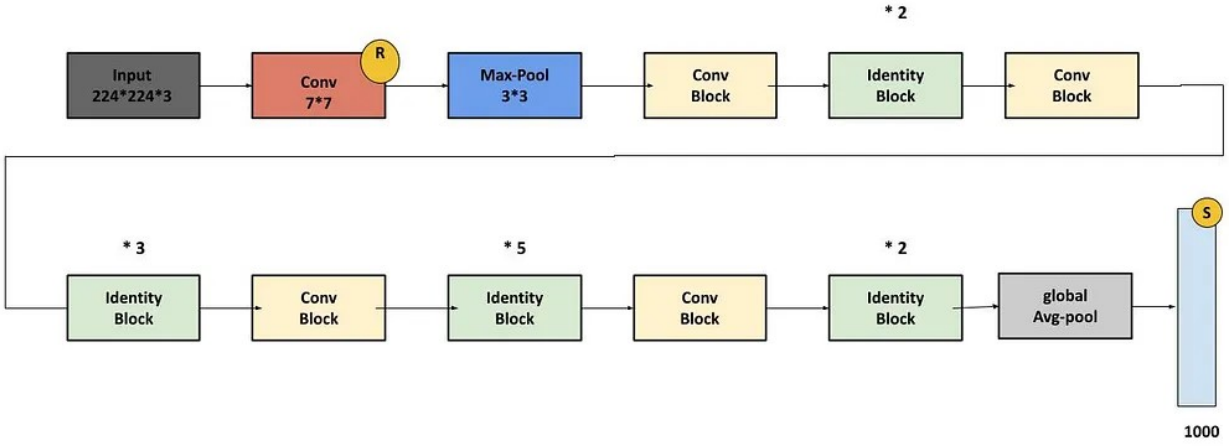
ResNet-50 :

Paper : Deep Residual Learning for Image Recognition.

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

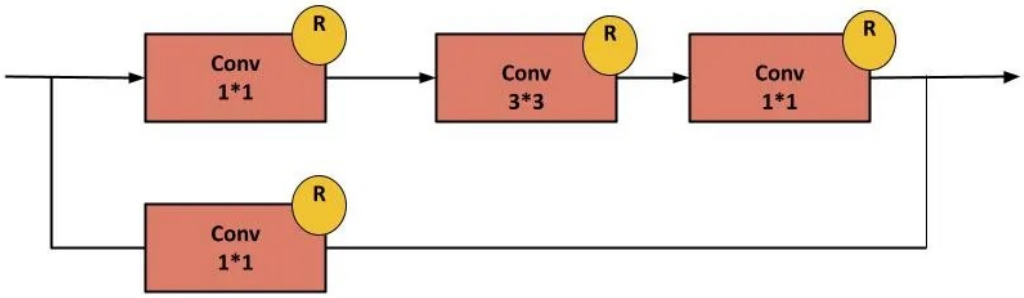
Published in : 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

The Main Architecture :



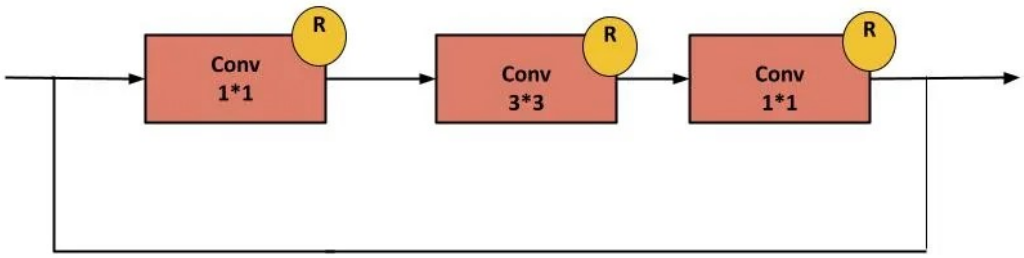
ResNet 50

Conv Block :



Conv Block

Identity Block :



identity Block

Implementation :

1. keras :

```
from keras.models import Model
from keras.layers.merge import concatenate
from keras.layers import Conv2D , MaxPool2D , Input ,AveragePooling2D, Dense , Dropout ,Activat



def IdentityBlock(prev_Layer , filters):
    f1 , f2 , f3 = filters

    x = Conv2D(filters=f1, kernel_size = (1,1) , strides=(1,1), padding='valid')(prev_Layer)
    x = BatchNormalization(axis=3)(x)
    x = Activation(activation='relu')(x)

    x = Conv2D(filters=f2, kernel_size = (3,3) , strides=(1,1), padding='same')(x)
    x = BatchNormalization(axis=3)(x)
    x = Activation(activation='relu')(x)

    x = Conv2D(filters=f3, kernel_size = (1,1) , strides=(1,1), padding='valid')(x)
    x = BatchNormalization(axis=3)(x)
    x = Activation(activation='relu')(x)

    x = concatenate([ x, prev_Layer ], axis=-1)
    x = Activation(activation='relu')(x)
    return x

def ConvBlock(prev_Layer , filters , strides):
    f1 , f2 , f3 = filters

    #Path 1
    x = Conv2D(filters=f1, kernel_size = (1,1) ,padding='valid', strides=strides)(prev_Layer)
    x = BatchNormalization(axis=3)(x)
    x = Activation(activation='relu')(x)

    x = Conv2D(filters=f2, kernel_size = (3,3) , padding='same' , strides=(1 ,1))(x)
    x = BatchNormalization(axis=3)(x)
    x = Activation(activation='relu')(x)

    x = Conv2D(filters=f3, kernel_size = (1,1), padding='valid' , strides=(1 ,1))(x)
    x = BatchNormalization(axis=3)(x)
    x = Activation(activation='relu')(x)

    #Path 2

    x2 = Conv2D(filters=f3, kernel_size=(1,1), padding='valid' , strides=strides)(prev_Layer)
    x2 = BatchNormalization(axis=3)(x2)

    x = concatenate([x , x2], axis=-1)
    x = Activation(activation='relu')(x)
    return x

def ResNet50():
    input_layer = Input(shape = (224, 224, 3))
    #Stage 1
```

```
    #Stage 1
    x = ZeroPadding2D((3, 3))(input_layer)
    x = Conv2D(filters = 64, kernel_size = (7,7), strides=(2,2)) (x)
    x = BatchNormalization(axis=3)(x)
    x = Activation(activation='relu')(x)
    x = MaxPool2D(pool_size=(3,3) , strides=(2,2))(x)

    #Stage 2
    x = ConvBlock(prev_Layer=x, filters = [64 , 64 , 256], strides = 1)
    x = IdentityBlock(prev_Layer=x, filters = [64,64,256])
    x = IdentityBlock(prev_Layer=x, filters = [64,64,256])

    #Stage 3
    x = ConvBlock(prev_Layer=x, filters = [128 , 128 , 512], strides = 2)
    x = IdentityBlock(prev_Layer=x, filters = [128 , 128 , 512])
    x = IdentityBlock(prev_Layer=x, filters = [128 , 128 , 512])
    x = IdentityBlock(prev_Layer=x, filters = [128 , 128 , 512])

    #Stage 4
    x = ConvBlock(prev_Layer=x, filters = [256 , 256 , 1024], strides = 2)
    x = IdentityBlock(prev_Layer=x, filters = [256 , 265 , 1024])
    x = IdentityBlock(prev_Layer=x, filters = [256 , 265 , 1024])
    x = IdentityBlock(prev_Layer=x, filters = [256 , 265 , 1024])
    x = IdentityBlock(prev_Layer=x, filters = [256 , 265 , 1024])
    x = IdentityBlock(prev_Layer=x, filters = [256 , 265 , 1024])

    #Stage 5
    x = ConvBlock(prev_Layer=x, filters = [512 , 512 , 2048], strides = 2)
    x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 2048])
    x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 2048])

    #Stage 6
    x = AveragePooling2D(pool_size=(7,7)) (x)

    x = Flatten()(x)
    x = Dense(units=1000, activation='softmax')(x)

    model = Model(inputs=input_layer , outputs = x , name='ResNet50')
    return model
```

2. PyTorch :

```python
import torch
import torch.nn as nn

class convolution2D(nn.Module):
  def __init__(self , in_channels , out_channels , kernel_size , stride , padding , activation
    super(convolution2D , self).__init__()
    self.conv = nn.Conv2d(in_channels , out_channels , kernel_size , stride , padding)
    self.batchNornalization = nn.BatchNorm2d(num_features=out_channels)
    self.activation = nn.ReLU()
    self.act = activation

  def forward(self , x):
    out = self.conv(x)
    out = self.batchNornalization(out)
    if self.act :
      out = self.activation(out)
    return out

class conv_Block(nn.Module):
  def __init__(self , in_channels , filters , stride):
    super(conv_Block , self).__init__()
    f1 , f2 , f3 = filters

    self.branch1 = nn.Sequential(
        convolution2D(in_channels , f1 , 1 , stride , 0 , True),
        convolution2D(f1 , f2 , 3 , 1 , 1, True),
        convolution2D(f2 , f3 , 1 , 1 , 0, False)
    )

    self.branch2 = convolution2D(in_channels , f3 , 1 , stride , 0, False)

    self.activation = nn.ReLU()

  def forward(self , x):

    branch1 = self.branch1(x)

    branch2 = self.branch2(x)

    out = torch.cat([branch1 , branch2] , 1)

    return self.activation(out)

class identity_Block(nn.Module):
  def __init__(self , in_channels , filters ):
    super(identity_Block , self).__init__()

    f1 , f2 , f3 = filters
    self.branch1 = nn.Sequential(
        convolution2D(in_channels , f1 , 1 , 1 , 0 , True),
        convolution2D(f1 , f2 , 3 , 1 , 1, True),
        convolution2D(f2 , f3 , 1 , 1 , 0, False)
```

```
        )

        self.activation = nn.ReLU()

    def forward(self , x):

        branch1 = self.branch1(x)

        branch2 = x

        out = torch.cat([branch1 , branch2] , 1)

        return self.activation(out)

class ResNet_50(nn.Module):
    def __init__(self):
        super(ResNet_50 , self).__init__()

        self.conv1 = convolution2D(in_channels = 3 , out_channels = 64, kernel_size = 7 , stride =
        self.MaxPooling1 = nn.MaxPool2d(kernel_size=3 , stride = 2)

        self.ConvBlock1 = conv_Block(64 , [64, 64,256] , 1)
        self.IdentityBlock1 = identity_Block(512 , [64,64,256])
        self.IdentityBlock2 = identity_Block(768 , [64,64,256])
        self.ConvBlock2 = conv_Block(1024 , [128, 128,512] , 2)
        self.IdentityBlock3 = identity_Block(1024 , [128,128,512])
        self.IdentityBlock4 = identity_Block(1536 , [128,128,512])
        self.IdentityBlock5 = identity_Block(2048 , [128,128,512])
        self.ConvBlock3 = conv_Block(2560 , [256, 256,1024] , 2)
        self.IdentityBlock6 = identity_Block(2048 , [256, 256,1024])
        self.IdentityBlock7 = identity_Block(3072 , [256, 256,1024])
        self.IdentityBlock8 = identity_Block(4096 , [256, 256,1024])
        self.IdentityBlock9 = identity_Block(5120 , [256, 256,1024])
        self.IdentityBlock10 = identity_Block(6144 , [256, 256,1024])
        self.ConvBlock4 = conv_Block(7168 , [512, 512,2048] , 2)
        self.IdentityBlock11 = identity_Block(4096 , [512, 512,2048])
        self.IdentityBlock12 = identity_Block(6144 , [512, 512,2048])
        self.globalAvgPooling = nn.Conv2d(in_channels=8192 , out_channels=8192 , kernel_size=7)
        self.fc = nn.Linear(in_features=8192 , out_features=1000)
        self.activation = nn.Softmax()

    def forward(self , x):

        out = self.conv1(x)
        out = self.MaxPooling1(out)

        out = self.ConvBlock1(out)

        out = self.IdentityBlock1(out)
        out = self.IdentityBlock2(out)

        out = self.ConvBlock2(out)

        out = self.IdentityBlock3(out)
        out = self.IdentityBlock4(out)
```

```
        out = self.IdentityBlock5(out)

        out = self.ConvBlock3(out)

        out = self.IdentityBlock6(out)
        out = self.IdentityBlock7(out)
        out = self.IdentityBlock8(out)
        out = self.IdentityBlock9(out)
        out = self.IdentityBlock10(out)

        out = self.ConvBlock4(out)

        out = self.IdentityBlock11(out)
        out = self.IdentityBlock12(out)

        out = self.globalAvgPooling(out)

        out = out.reshape(out.shape[0] , -1)

        out = self.fc(out)
        out = self.activation(out)

        return out
```

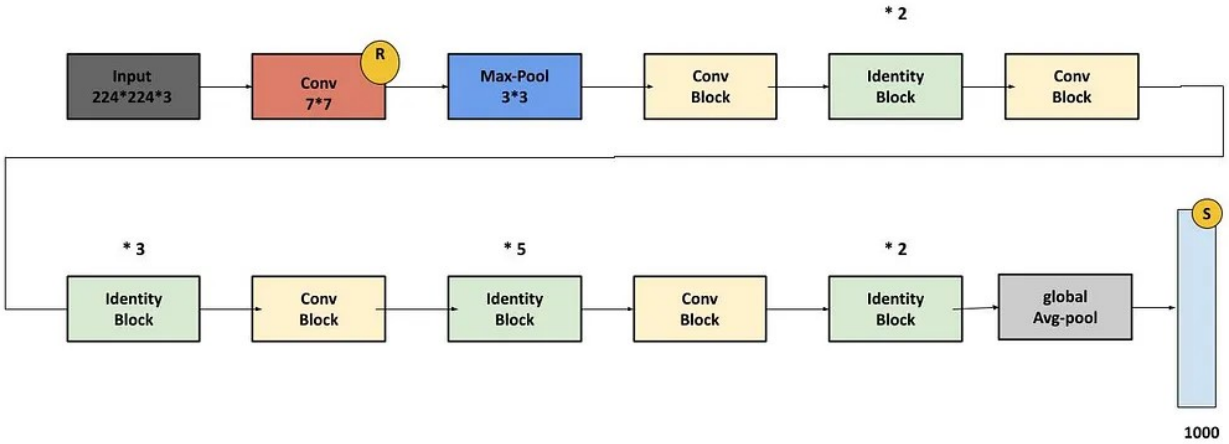
ResNeXt-50 :

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 .

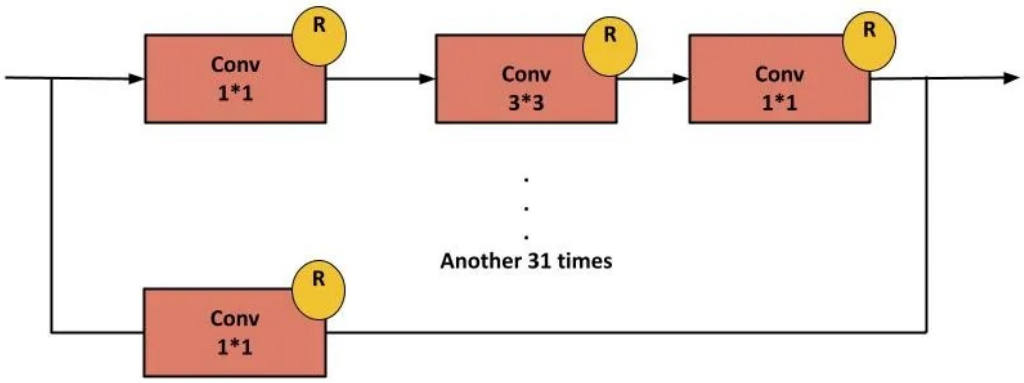
Published in : 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) .

The Main Architecture :



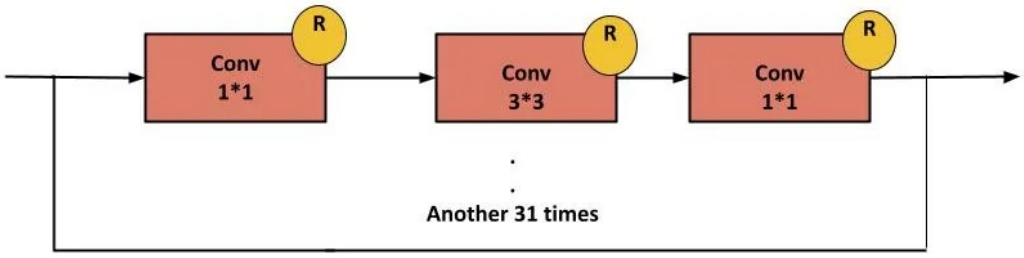
ResNeXt-50

Conv Block :



Conv Block

Identity Block :



identity Block

Implementation :

1. Keras :

```
from keras.models import Model
from keras.layers import Conv2D , MaxPool2D , ZeroPadding2D, Input ,AveragePooling2D, Dense , D
from keras.layers import Add

def IdentityBlock(prev_Layer , filters):

    f1 , f2 ,f3 = filters
    block = []

    for i in range(32):
        x = Conv2D(filters=f1, kernel_size = (1,1) , strides=(1,1), padding='valid')(prev_Layer
        x = BatchNormalization(axis=3)(x)
        x = Activation(activation='relu')(x)

        x = Conv2D(filters=f2, kernel_size = (3,3) , strides=(1,1), padding='same')(x)
        x = BatchNormalization(axis=3)(x)
        x = Activation(activation='relu')(x)

        x = Conv2D(filters=f3, kernel_size = (1,1) , strides=(1,1), padding='valid')(x)
        x = BatchNormalization(axis=3)(x)
        x = Activation(activation='relu')(x)
        block.append(x)

    block.append(prev_Layer)
    x = Add()(block)
    x = Activation(activation='relu')(x)

    return x


def ConvBlock(prev_Layer , filters , strides):
    f1 , f2 , f3 = filters

    block = []

    for i in range(32):
        x = Conv2D(filters=f1, kernel_size = (1,1) ,padding='valid', strides=strides)(prev_Laye
        x = BatchNormalization(axis=3)(x)
        x = Activation(activation='relu')(x)

        x = Conv2D(filters=f2, kernel_size = (3,3) , padding='same' , strides=(1 ,1))(x)
        x = BatchNormalization(axis=3)(x)
        x = Activation(activation='relu')(x)

        x = Conv2D(filters=f3, kernel_size = (1,1), padding='valid' , strides=(1 ,1))(x)
        x = BatchNormalization(axis=3)(x)
        x = Activation(activation='relu')(x)
        block.append(x)

    x2 = Conv2D(filters=f3, kernel_size=(1,1), padding='valid' , strides=strides)(prev_Layer)
    x2 = BatchNormalization(axis=3)(x2)

    block.append(x2)
```

```
    block.append(x2)
    x = Add()(block)
    x = Activation(activation='relu')(x)
    return x


def ResNext():
    input_layer = Input(shape = (224, 224, 3))
    #Stage 1
    x = ZeroPadding2D((3, 3))(input_layer)
    x = Conv2D(filters = 64, kernel_size = (7,7), strides=(2,2)) (x)
    x = BatchNormalization(axis=3)(x)
    x = Activation(activation='relu')(x)
    x = MaxPool2D(pool_size=(3,3) , strides=(2,2))(x)

    #Stage 2
    x = ConvBlock(prev_Layer=x, filters = [128 , 128 , 256], strides = 1)
    x = IdentityBlock(prev_Layer=x, filters = [128 , 128 , 256])
    x = IdentityBlock(prev_Layer=x, filters = [128 , 128 , 256])

    #Stage 3
    x = ConvBlock(prev_Layer=x, filters = [256 , 256 , 512], strides = 2)
    x = IdentityBlock(prev_Layer=x, filters = [256 , 256 , 512])
    x = IdentityBlock(prev_Layer=x, filters = [256 , 256 , 512])
    x = IdentityBlock(prev_Layer=x, filters = [256 , 256 , 512])

    #Stage 4
    x = ConvBlock(prev_Layer=x, filters = [512 , 512 , 1024], strides = 2)
    x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 1024])
    x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 1024])
    x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 1024])
    x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 1024])
    x = IdentityBlock(prev_Layer=x, filters = [512 , 512 , 1024])

    #Stage 5
    x = ConvBlock(prev_Layer=x, filters = [1024 , 1024 , 2048], strides = 2)
    x = IdentityBlock(prev_Layer=x, filters = [1024 , 1024 , 2048])
    x = IdentityBlock(prev_Layer=x, filters = [1024 , 1024 , 2048])

    #Stage 6
    x = AveragePooling2D(pool_size=(7,7)) (x)

    x = Flatten()(x)
    x = Dense(units=1000, activation='softmax')(x)

    model = Model(inputs=input_layer , outputs = x , name='ResNet50')
    return model
```

2. PyTorch :

```python
import torch
import torch.nn as nn
from torchsummary import summary

class convolution2D(nn.Module):
  def __init__(self , in_channels , out_channels , kernel_size , stride , padding , activation
    super(convolution2D , self).__init__()
    self.conv = nn.Conv2d(in_channels , out_channels , kernel_size , stride , padding)
    self.batchNornalization = nn.BatchNorm2d(num_features=out_channels)
    self.activation = nn.ReLU()
    self.act = activation

  def forward(self , x):
    out = self.conv(x)
    out = self.batchNornalization(out)
    if self.act :
      out = self.activation(out)
    return out

class Conv_Block(nn.Module):

  def __init__(self , in_channels , filters , stride):
    super(Conv_Block , self).__init__()
    f1 , f2 , f3 = filters

    self.branch1 = nn.Sequential(
        convolution2D(in_channels , f1 , 1 , stride , 0 , True),
        convolution2D(f1 , f2 , 3 , 1 , 1, True),
        convolution2D(f2 , f3 , 1 , 1 , 0, False)
    )

    self.branch2 = convolution2D(in_channels , f3 , 1 , 1 , 0, False)
    self.activation = nn.ReLU()

  def forward(self,x):

    paths = [self.branch1(x) for i in range(32)]
    paths.append(self.branch2(x))

    out = torch.cat(paths , 1)
    return self.activation(out)

class Identity_Block(nn.Module):

  def __init__(self , in_channels , filters):
    super(Identity_Block , self).__init__()
    f1 , f2 , f3 = filters

    self.branch1 = nn.Sequential(
        convolution2D(in_channels , f1 , 1 , 1 , 0 , True),
        convolution2D(f1 , f2 , 3 , 1 , 1, True),
        convolution2D(f2 , f3 , 1 , 1 , 0, False)
```

```
        )
        self.activation = nn.ReLU()
    def forward(self , x):
        paths = [self.branch1(x) for i in range(32)]
        paths.append(x)

        out = torch.cat(paths , 1)
        return self.activation(out)

class ResNext_50(nn.Module):

    def __init__(self):
        super(ResNext_50,self).__init__()
        self.conv1 = convolution2D(in_channels = 3 , out_channels = 64, kernel_size = 7 , stride =
        self.MaxPooling1 = nn.MaxPool2d(kernel_size=3 , stride = 2)

        self.convBlock1 = Conv_Block(64 , [128 , 128 , 256] , 1)
        self.IdentityBlock1 = Identity_Block(8448 , [128 , 128 , 256])
        self.IdentityBlock2 = Identity_Block(16640 , [128 , 128 , 256])
        self.convBlock2 = Conv_Block(16896 , [256 , 256 , 512] , 2)
        self.IdentityBlock3 = Identity_Block(33280 , [256 , 256 , 512])
        self.IdentityBlock4 = Identity_Block(49664 , [256 , 256 , 512])
        self.IdentityBlock5 = Identity_Block(66048 , [256 , 256 , 512])
        self.convBlock3 = Conv_Block(33792 , [512 , 512 , 1024] , 2)
        self.IdentityBlock6 = Identity_Block(66560 , [512 , 512 , 1024])
        self.IdentityBlock7 = Identity_Block(99328 , [512 , 512 , 1024])
        self.IdentityBlock8 = Identity_Block(132096 , [512 , 512 , 1024])
        self.IdentityBlock9 = Identity_Block(164864, [512 , 512 , 1024])
        self.IdentityBlock10 = Identity_Block(197632 , [512 , 512 , 1024])
        self.convBlock4 = Conv_Block(67584 , [1024 , 1024 , 2048] , 2)
        self.IdentityBlock11 = Identity_Block(133120, [1024 , 1024 , 2048])
        self.IdentityBlock12 = Identity_Block(192656, [1024 , 1024 , 2048])
        self.globalAvgPooling = nn.Conv2d(in_channels=8192 , out_channels=8192 , kernel_size=7)
        self.fc = nn.Linear(in_features=192656 , out_features=1000)
        self.activation = nn.Softmax()
    def forward(self , x):
        out = self.conv1(x)
        out = self.MaxPooling1(out)

        out = self.convBlock1(out)

        out = self.IdentityBlock1(out)
        out = self.IdentityBlock2(out)

        out = self.convBlock2(out)

        out = self.IdentityBlock3(out)
        out = self.IdentityBlock4(out)
        out = self.IdentityBlock5(out)

        out = self.convBlock3(out)

        out = self.IdentityBlock6(out)
        out = self.IdentityBlock7(out)
        out = self.IdentityBlock8(out)
```

```
        out = self.IdentityBlock9(out)
        out = self.IdentityBlock10(out)

        out = self.convBlock4(out)

        out = self.IdentityBlock11(out)
        out = self.IdentityBlock12(out)

        out = self.globalAvgPooling(out)

        out = out.reshape(out.shape[0] , -1)

        out = self.fc(out)
        out = self.activation(out)
        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

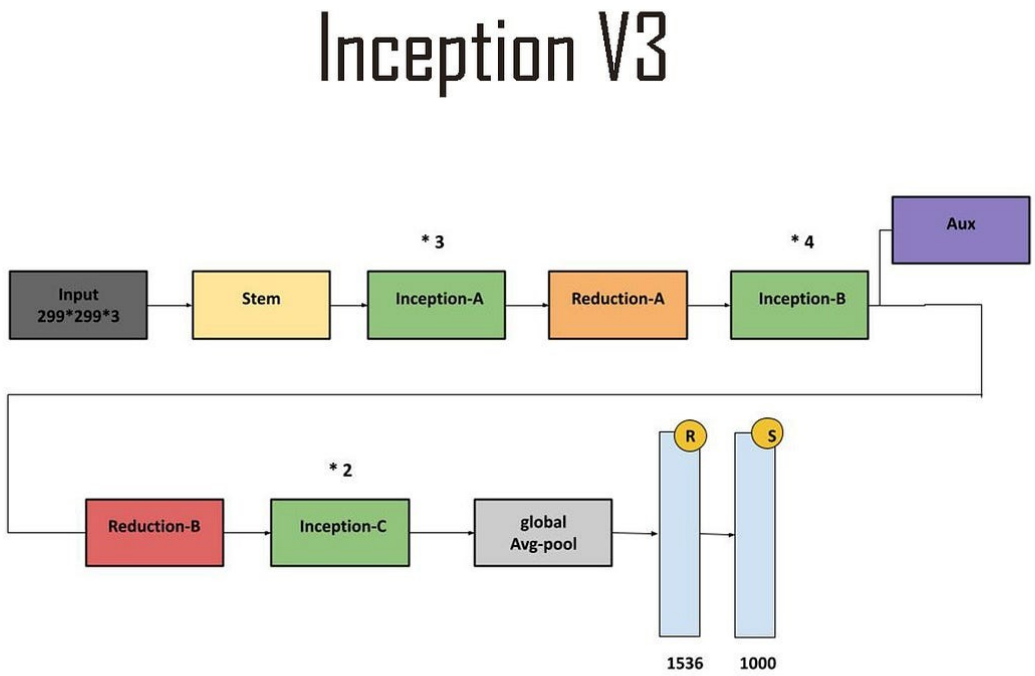
Follow

Written by Anas BRITAL

60 Followers

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

More from Anas BRITAL



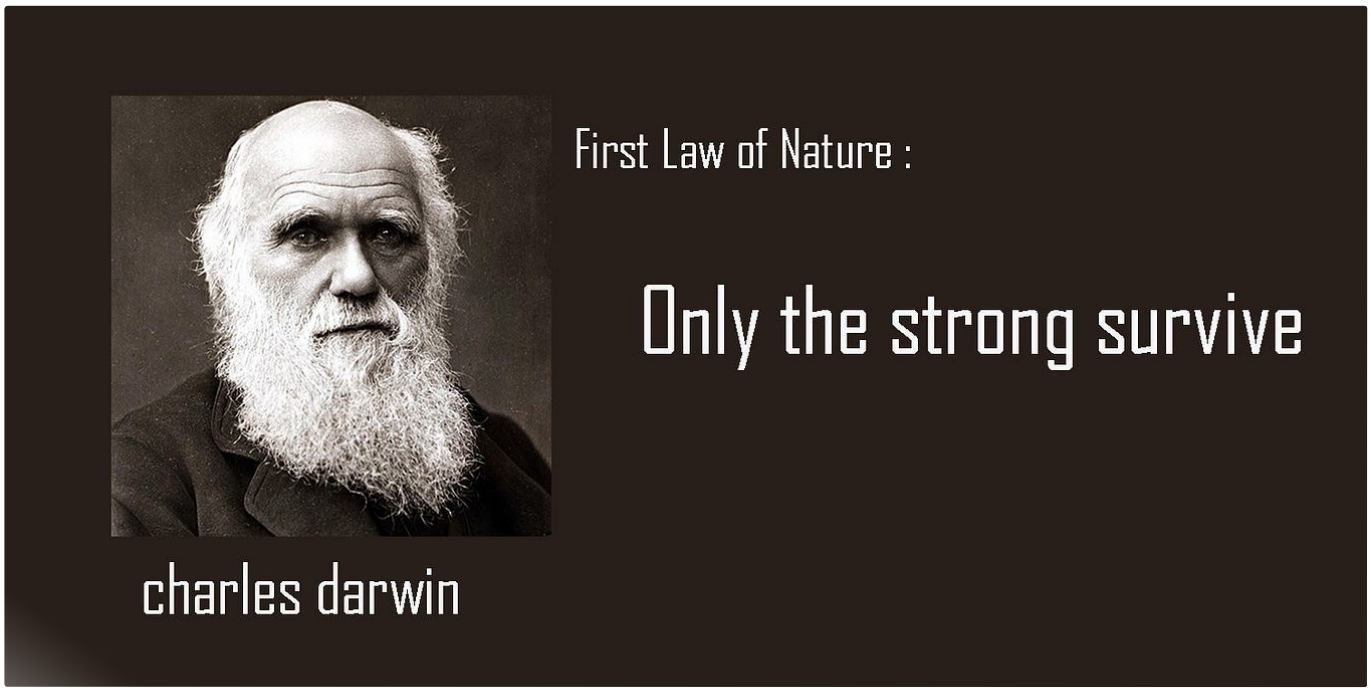
Anas BRITAL

Inception V3 CNN Architecture Explained .

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

4 min read · Oct 23, 2021

12



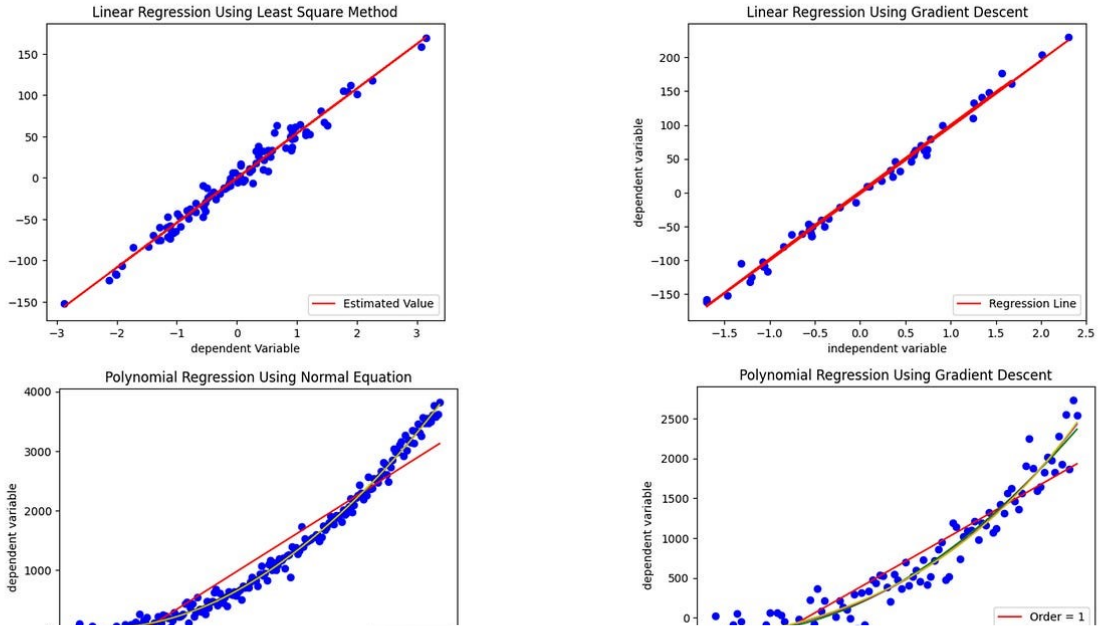
Anas BRITAL

Genetic Algorithm Explained :

Everything you need to know About Genetic Algorithm .

10 min read · Oct 16, 2021

113 2



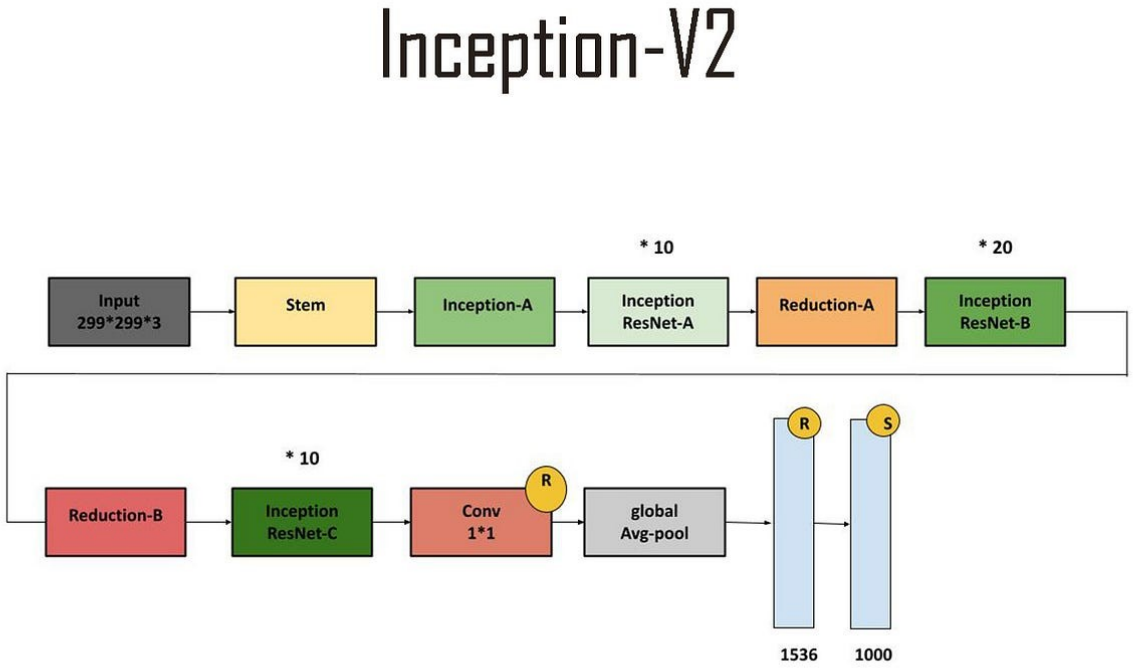
Anas BRITAL

Regression Analysis

Regression analysis models Explained and Implemented Using Python

5 min read · Nov 6, 2021

55



Anas BRITAL

Inception V2 CNN Architecture Explained .

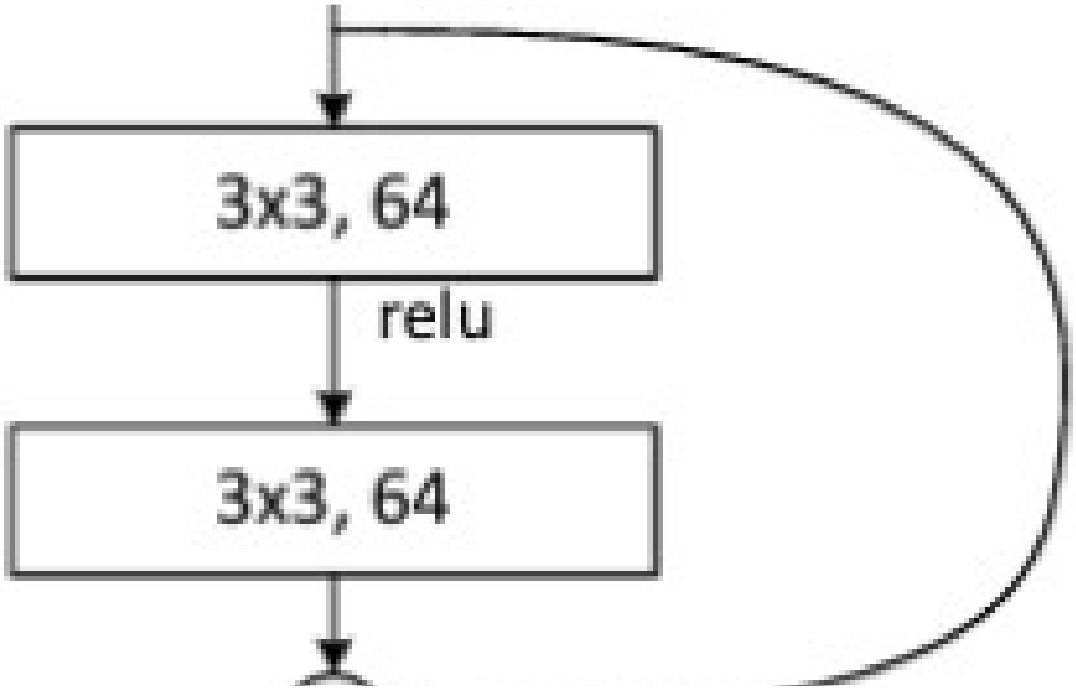
Inception-V2 CNN Architecture illustrated and Implemented in both Keras and PyTorch .

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...

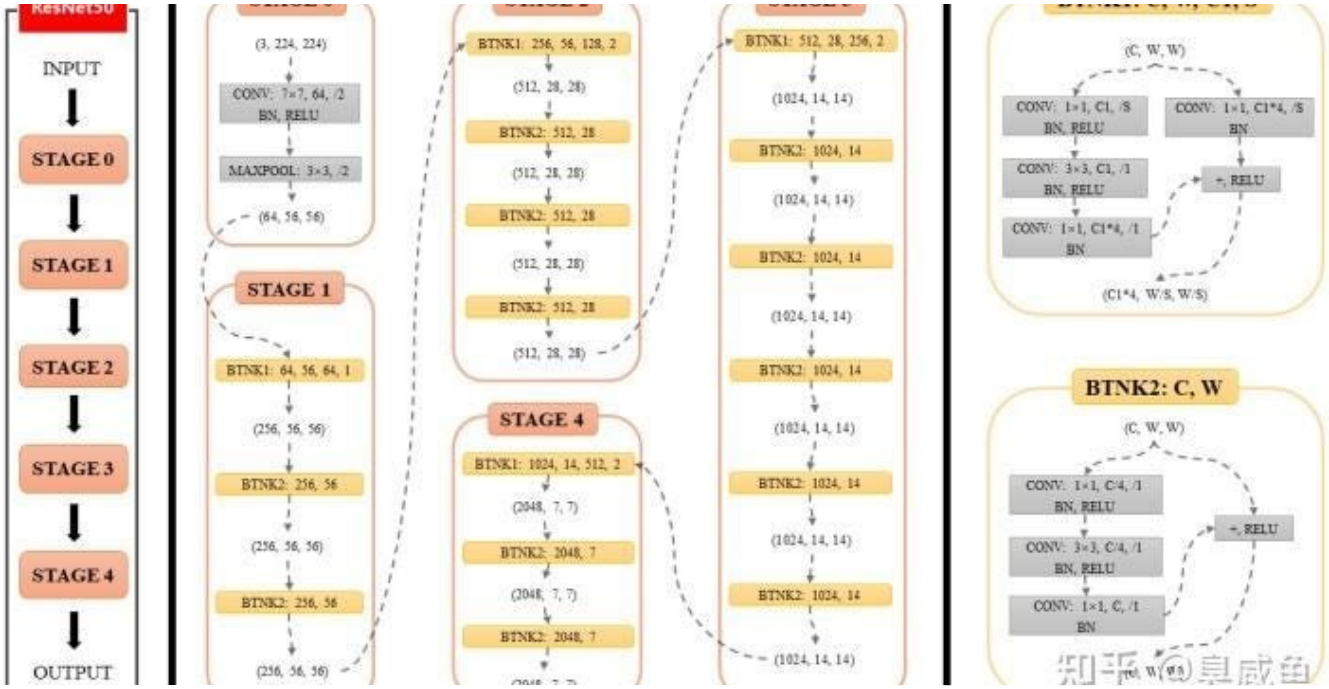
6 min read · Nov 14, 2023

102

Lists

Practical Guides to Machine Learning
10 stories · 1312 saves

Natural Language Processing
1369 stories · 865 saves

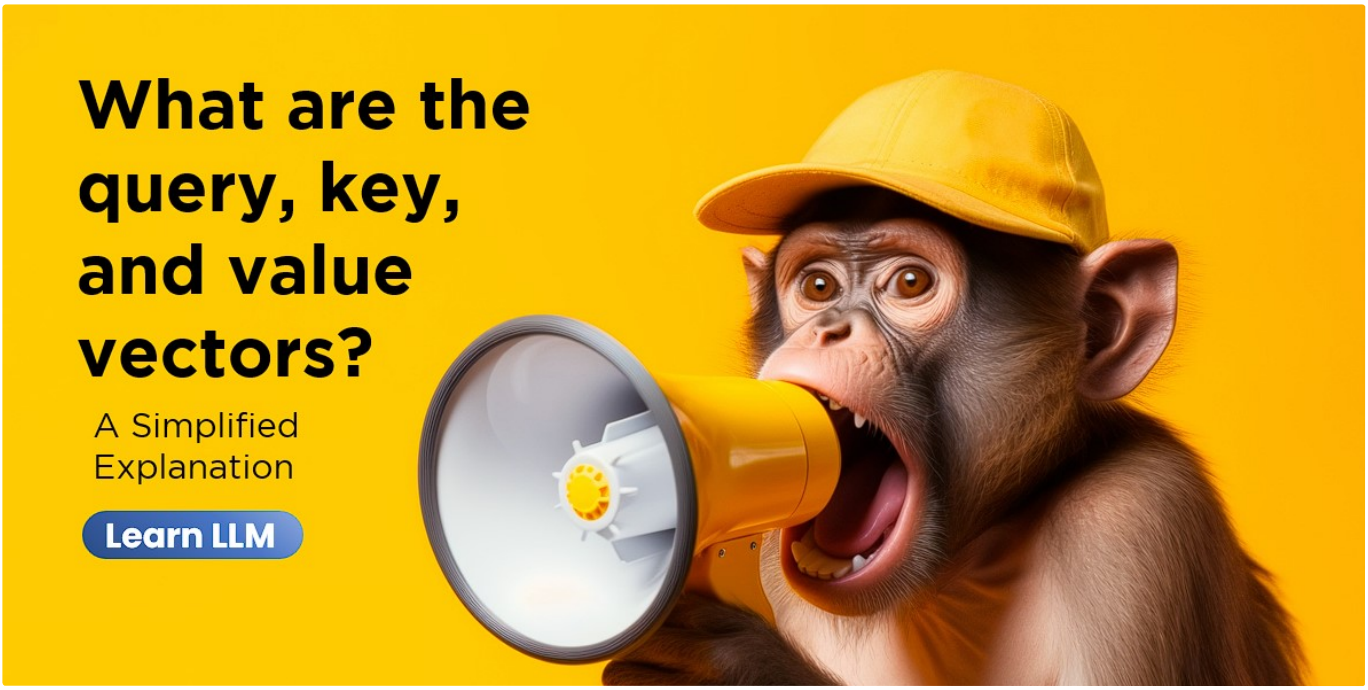


noplaxochia

Pytorch ResNet from Scratch

Pytorch implementation of

1 min read · Jan 15, 2024



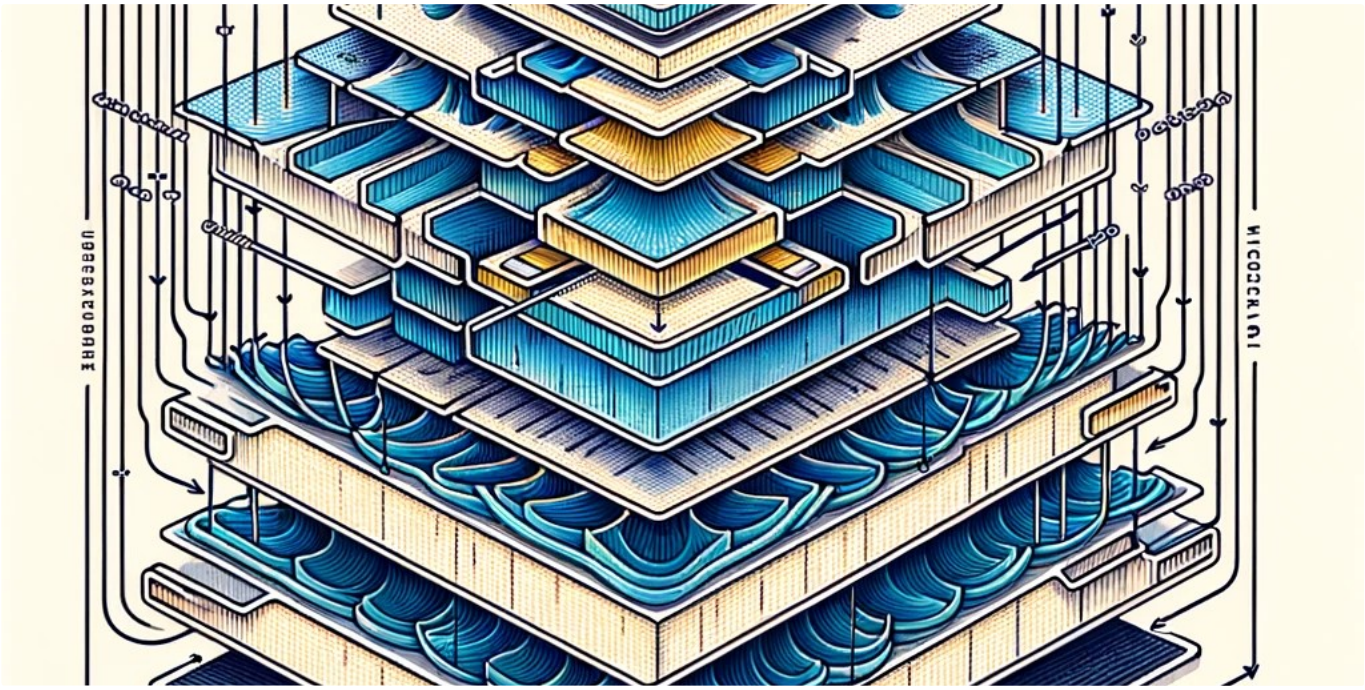
RAHULRAJ P V

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



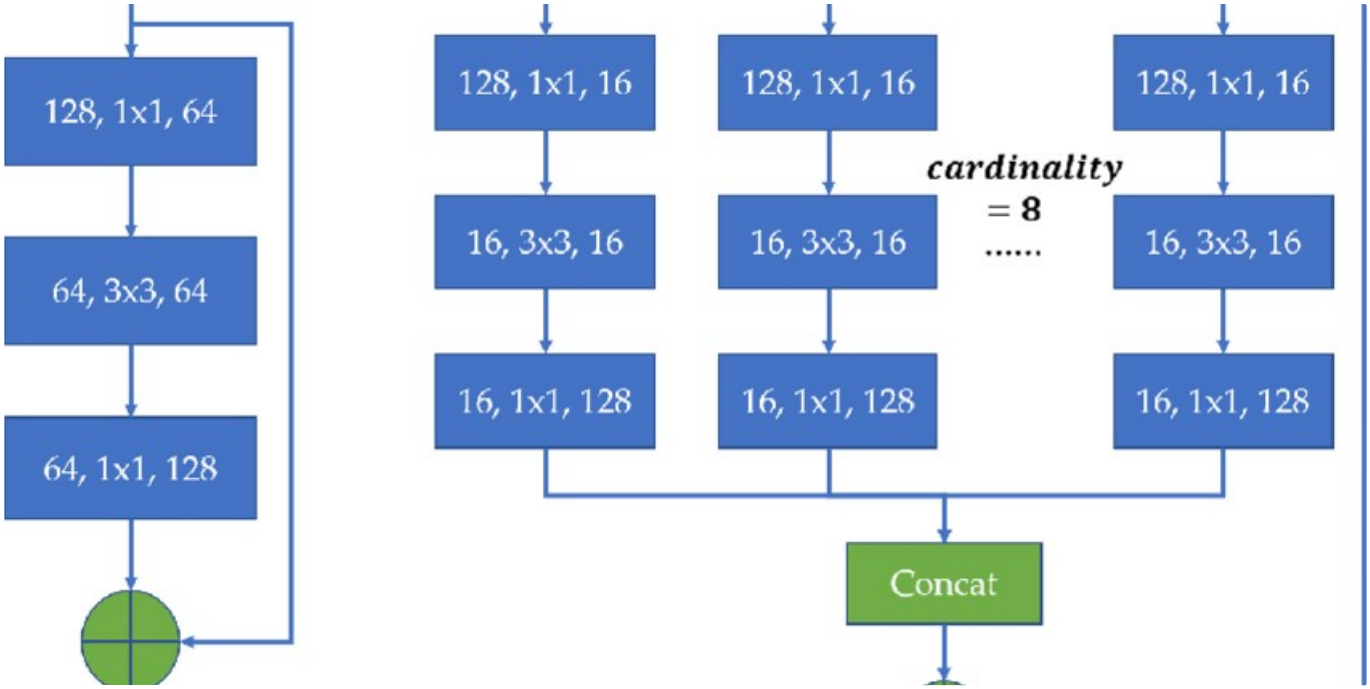
Bragadeesh Sundararajan

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