Detailed Explanation of Resnet CNN Model.

TANISH SHARMA · Follow

10 min read · Mar 6, 2023

ResNets or Residual networks are a type of deep convolutional neural network architecture that was first introduced in Dec 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun @ MSRA(Microsoft Asia). Won 1st place in the ILSVRC 2015 classification competition Won 1st Place in ImageNet Classification: “Ultra-deep”-152 layer nets. The extremely deep representations also have excellent generalization performance on other recognition tasks and lead them to further win 1st place on ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation in the same ILSVRC & COCO 2015 competitions.

Why ResNets Needed?

In traditional deep neural networks or plain deep neural networks, the vanishing gradient problem occurs means that gradients become very small as they are propagated through many layers of a neural network. In simple words as the no of layers increases training or test error gets worst. This can

lead to slower learning and even a complete lack of learning in very deep architectures. ResNets solve this problem by introducing skip connections, also known as residual connections, which allow the gradient to bypass certain layers in the network.

When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error, as reported and thoroughly verified by researchers in their experiments. The degradation (of training accuracy) indicates that not all systems are similarly easy to optimize. To overcome this we use residual/ identity mapping(block). It means we skip the blocks of convolution that exist between two Relu Activation units, feeding the o/p of the first relu to 2nd relu. By adding this residual connection, the network is able to learn the residual function instead of directly learning the underlying mapping. This can lead to more efficient learning and improved performance, especially in very deep architectures. Adding an additional layer would not hurt performance as regularization/ weights skip over then, this is a guarantee that performance will not decrease as you grow layers from 20 to 50 or 50 to 100 just simply add an identity block.

As we know if x==non-zero: Relu(x)= Relu(Relu(x))

else: Relu(x)=0

If the weight layers are useless then regularization becomes zero so then to skip that connection would be helpful. if useful then adding those weight layers increase the performance.

Image Source Credits: Actual Research paper

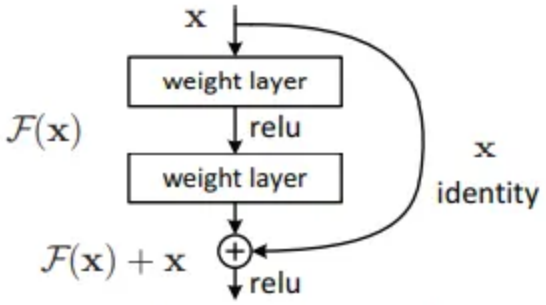


Figure 2. Residual learning: a building block.

X=input, directly adding x to F(x).

The inspiration behind ResNets is to learn residual functions instead of directly learning the desired underlying mapping between the input and output. This is achieved by adding skip connections that bypass one or more layers in the network, allowing the input to be added directly to the output of a later layer. The residual function is defined as the difference between the input and output of the layer.

Types of ResNets:

1. ResNet18, 2. ResNet34, 3.ResNet50, 4. ResNet101, 5. ResNet152

Suffix number refers to the number of layers used in their network architecture. At this point, we'll cover the architecture of ResNet50.

What is ResNet 50?

One of the most well-known ResNet architectures is ResNet50, which consists of 50 layers and achieved state-of-the-art performance on the ImageNet dataset in 2015. ResNet50 consists of 16 residual blocks, with each block consisting of several convolutional layers with residual connections. The architecture also includes pooling layers, fully connected layers, and a softmax output layer for classification.

Architecture of ResNet50

Input layer: The input layer of ResNet50 takes an image of size 224 x 224 x 3 as input. The 3 represents the RGB color channels of the image.

Components used in this architecture:

1. **Convolutional layers:** Convolutional layers are the building blocks of convolutional neural networks (CNNs), a type of deep learning algorithm that is commonly used for image processing tasks. Convolutional layers are designed to extract features from the input image by applying a series of convolutional filters to the image. Each convolutional filter is a small matrix of weights that is applied to a local region of the input image. The filter slides over the entire image, producing a feature map that

highlights patterns and edges in the image. The weights of the filter are learned during the training process, allowing the network to automatically learn the best set of filters for the given task. By applying a series of convolutional filters to the input image, convolutional layers are able to extract features from the image and produce a set of feature maps that can be used as input to the next layer of the network.

2. **Batch Normalization**: Batch normalization is typically applied after the convolutional or fully connected layers in a neural network but before the activation function. It is a widely used technique in deep learning and has been shown to improve the performance of many types of neural networks. Faster convergence, Improved generalization, and Regularization are some benefits of batch normalization.

3. **Relu Activation**: It is an activation function commonly used in neural networks. It is a simple and effective way to introduce nonlinearity into the output of a neuron. The function is defined as: ReLU(x) = max(0, x). In other words, the output of the ReLU activation function is equal to the input if the input is positive, and 0 if the input is negative. This means that ReLU is a piecewise linear function, which makes it computationally efficient to compute and differentiate and able to prevent the vanishing gradient problem.

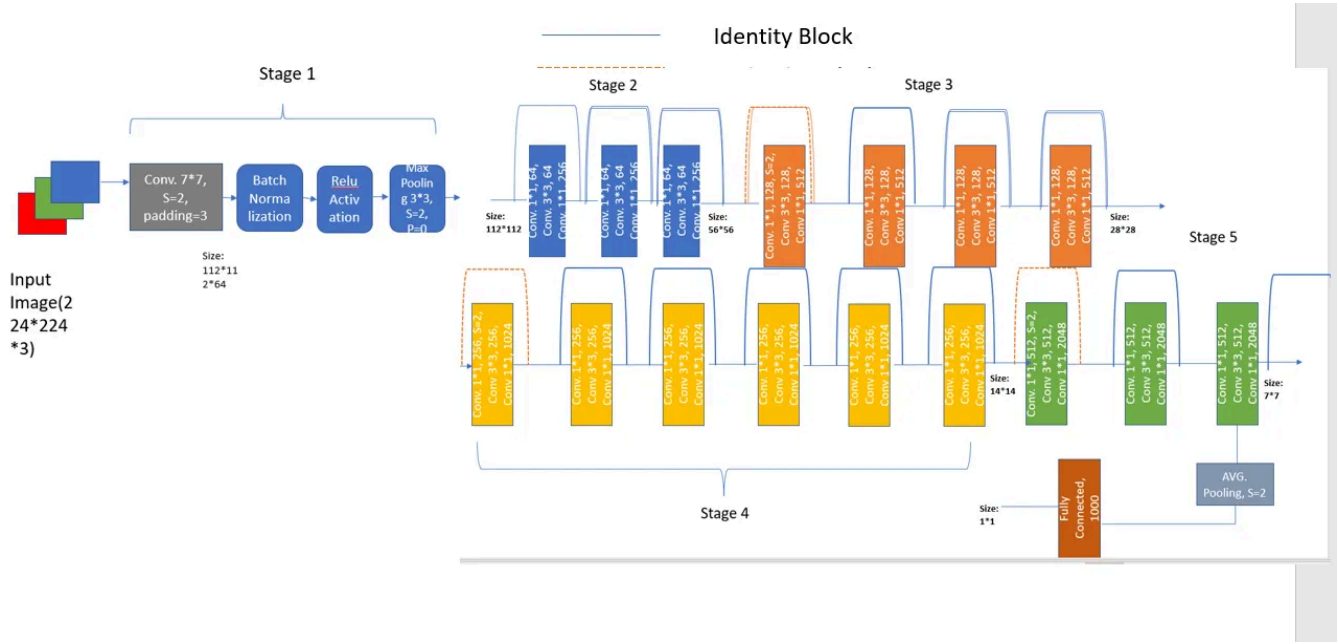
4. **Max Pooling**: It is used to reduce the spatial dimensions of the feature maps produced by the convolutional layers while retaining the most important information. Which can help to reduce the computational cost of the network and prevent overfitting. By selecting the maximum value in each window, the operation retains the most important features in the input feature map, while reducing the noise and the effect of small variations.

6. **Fully Connected Layers:** Also known as dense layers are a type of neural network layer where all neurons in the layer are connected to every neuron in the previous layer. These layers are typically used as the final layers in a neural network and are responsible for making the final predictions, where each neuron in the layer receives inputs from all neurons in the previous layer. The output of each neuron is then computed as a weighted sum of these inputs, followed by an activation function.
7. **Identity Block:** An identity block is the basic building block of the ResNet50 architecture, and it is used to maintain the same dimension of the input and output. An identity block consists of three convolutional layers, each followed by batch normalization and ReLU activation functions. The input is added to the output of the third convolutional layer.
8. **Global Average Pooling:** Reduce the spatial dimensions of the output tensor to a vector by applying global average pooling. This operation computes the average of each feature map, resulting in a feature vector with the same number of channels as the number of filters in the last convolutional layer.
9. **Projection block:** A projection block is used when the input and output dimensions are different. This block includes a convolutional layer with a stride of (2, 2) to downsample the input, and a convolutional layer with a (1, 1) filter size to change the depth of the input to match the output.

Stage 1: first convolutional layer, which takes as input the raw image with dimensions (224, 224, 3) and outputs a feature map with dimensions (112,

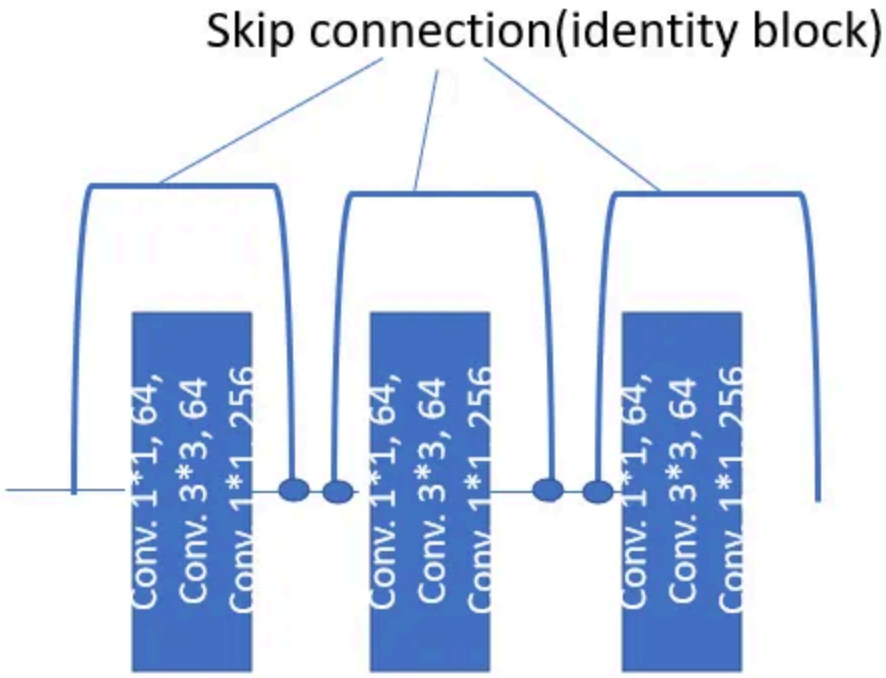
112, 64). This layer uses 64 filters, each with a kernel size of (7, 7) and a stride of (2, 2) to downsample the input image by a factor of 2 in both the width and height dimensions. After the convolutional layer, a batch normalization layer is applied to normalize the activations, followed by a ReLU activation function to introduce nonlinearity. Finally, a max pooling layer with a pool size of (3, 3) and a stride of (2, 2) is applied to further downsample the feature map by a factor of 2 in both dimensions.

This produces a feature map with dimensions (56, 56, 64), which serves as the input to the subsequent convolutional blocks in the ResNet50 architecture.

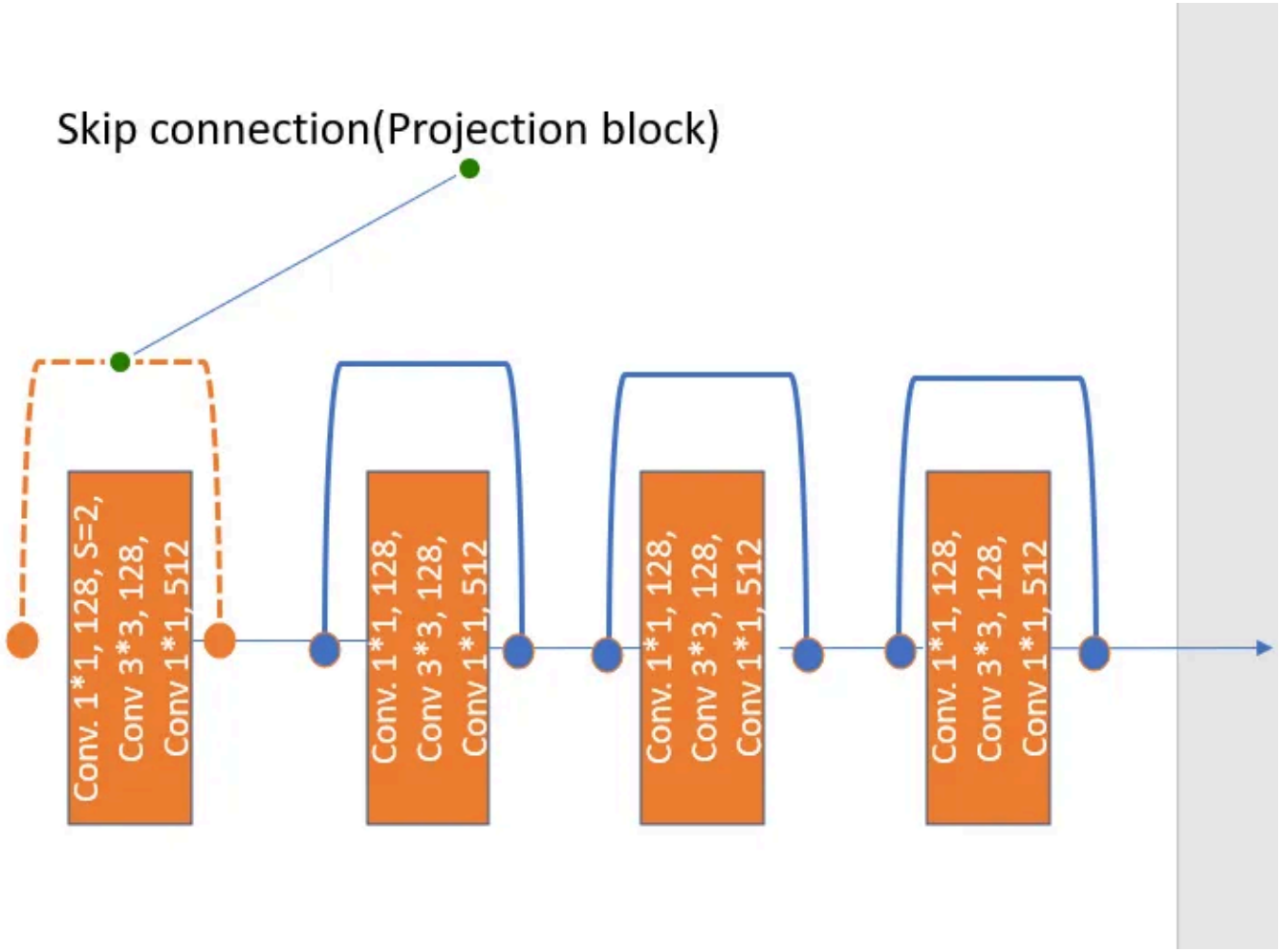


Stage 2: In this uses 9 Layers of convolutional layer followed by batch normalization and relu activation. As per above, you can see the identity block(connecting blue color lines from one to another block of convolutions). The output of the first stage is forwarded to the first layer of convolution with 64 kernel filters and a size of 1*1 o/p of this feed to 3*3 convolution size with the same 64 kernel filters, o/p this feed to conv. 1*1

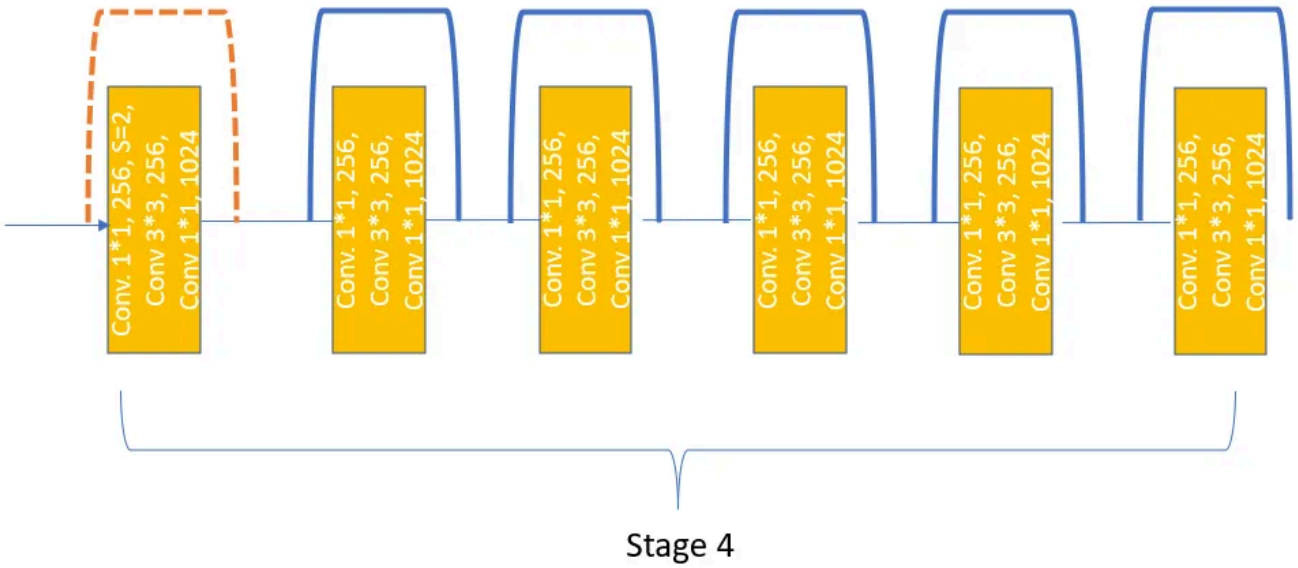
with 256 kernel filters. Now along with this, we create a skip connection/ residual connection, in which we pass the input(output from the max pool at stage 1) with size of image 56*56 to relu activation before the next convolution layers.



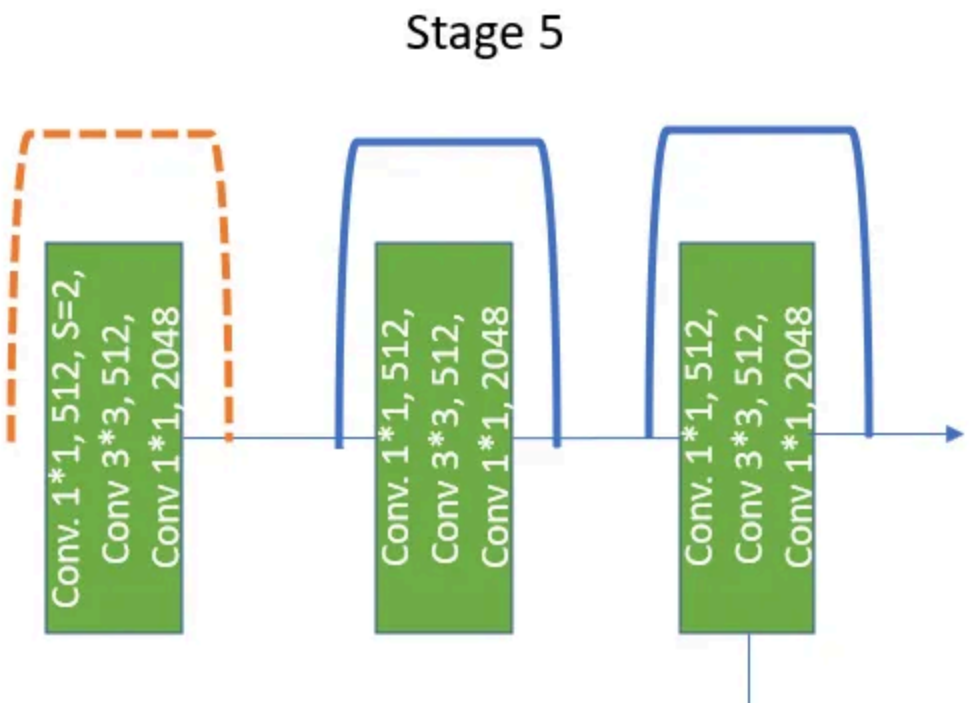
Stage 3: In this, 12 layers of convolutional layer followed by batch normalization and relu activation are used. Here Projection block used as the dimension of input and output is not the same, bypassing the first three layers convolutional layer, in the first layer convoluting with the size of kernel 1*1 with 128 filters (stride=2)to downsample the input and to change the depth of the input to match the output followed by convolution of 3*3, with, a same number of filters and conv. 1*1, kernel filters=512. Further use 9 layers of convolution with 1*1, filters=128=>3*3, filters=128 =>1*1, filters=512. The output of these 12th layers with size of image 28*28 forwards to relu activation before stage 4.



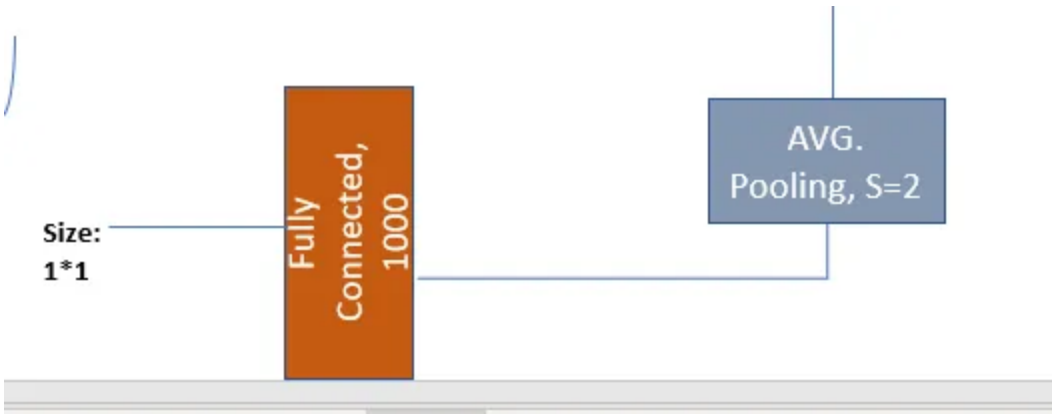
Stage4: In this stage uses 18 layers of convolution each followed by batch normalization and relu activation function. Output from relu activation has been forwarded to the first layer of the fourth stage where the dimension of image size is different so use projection block here in which input is forwarded to the convolution block to match the size of i/p with o/p at the second conv. block. The rest of the process remains the same as like previous stage convoluting layer by layer to understand more deep features of the image.



Stage 5: In this stage uses 9 layers of convolution each followed by batch normalization and relu activation function. Output from relu activation has been forwarded to with size of image 14*14 the first layer of the 5th stage where the dimension of image size is different so use projection block here in which input is forwarded to the convolution block to match the size of i/p with o/p at the second conv. block. The very first layer of the first convolutional block reducing the size of the image using 512 kernel filters with size of 1*1(stride=2) followed by conv 3*3 with 512 filters and conv. 1*1 with 2048 filters.



Finally, the output of the last relu activation unit at the fifth stage will be passed on to Average Pooling which reduces the spatial dimensions of the output tensor to a vector by applying global average pooling with stride size=2. This operation computes the average of each feature map, resulting in a feature vector with the same number of channels as the number of filters in the last convolutional layer.



Can i get the Code?

Here it is sample code of resnet50 in keras. Source credit C**hatGPT**.

1. Defining convolutional block function, in which input is processed with conv. followed by batch normalization and relu activation.

```
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Activatio
from tensorflow.keras import Model

def conv_block(x, filters, kernel_size, strides, padding='same'):
    x = Conv2D(filters=filters, kernel_size=kernel_size, strides=strides, paddin
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    return x
```

2. Identity block and Projection block function:

As we know identity block used when the dimension of o/p and i/p are same. In this line, we just adding the input directly to the activation unit relu by passing the convolution layers.

But in projection block, as dimensions are different we have to make them same we use another convolution of 1*1 on o/p of previous stage in skip connection.

```
def identity_block(x, filters):
    shortcut = x
    x = conv_block(x, filters=filters, kernel_size=(1, 1), strides=(1, 1))
    x = conv_block(x, filters=filters, kernel_size=(3, 3), strides=(1, 1))
    x = Conv2D(filters=filters * 4, kernel_size=(1, 1))(x)
    x = BatchNormalization()(x)
    x = Add()([x, shortcut])
    x = Activation('relu')(x)
    return x
```

```
def projection_block(x, filters, strides):
    shortcut = x
    x = conv_block(x, filters=filters, kernel_size=(1, 1), strides=strides)
    x = conv_block(x, filters=filters, kernel_size=(3, 3), strides=(1, 1))
    x = Conv2D(filters=filters * 4, kernel_size=(1, 1))(x)
    x = BatchNormalization()(x)
    shortcut = Conv2D(filters=filters * 4, kernel_size=(1, 1), strides=strides)(
    shortcut = BatchNormalization()(shortcut)
    x = Add()([x, shortcut])
    x = Activation('relu')(x)
    return x
```

3. Main resnet 50 function: Here we call first conv_block function passing input with kernel (7,7) and 64 such filters

```
def ResNet50(input_shape=(224, 224, 3), num_classes=1000):
    inputs = Input(shape=input_shape)

    # initial conv layer
    x = conv_block(inputs, filters=64, kernel_size=(7, 7), strides=(2, 2), paddi
    x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same')(x)

    # conv block 1
    x = projection_block(x, filters=64, strides=(1, 1))
    x = identity_block(x, filters=64)
    x = identity_block(x, filters=64)

    # conv block 2
    x = projection_block(x, filters=128, strides=(2, 2))
    x = identity_block(x, filters=128)
    x = identity_block(x, filters=128)
    x = identity_block(x, filters=128)

    # conv block 3
    x = projection_block(x, filters=256, strides=(2, 2))
    x = identity_block(x, filters=256)
    x = identity_block(x, filters=256)
    x = identity_block(x, filters=256)
    x = identity_block(x, filters=256)
    x = identity_block(x, filters=256)

    # conv block 4
```

```
    x = projection_block(x, filters=512, strides=(2, 2))
    x = identity_block(x, filters=512)
    x = identity_block(x, filters=512)

    # global average pooling and dense layer
    x = GlobalAveragePooling2D()(x)
    outputs = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=inputs, outputs=outputs)
    return model
```

Deep Learning | Image Recognition | Resnet50 | Python | Computer Vision

Written by TANISH SHARMA

31 Followers · 17 Following

Follow

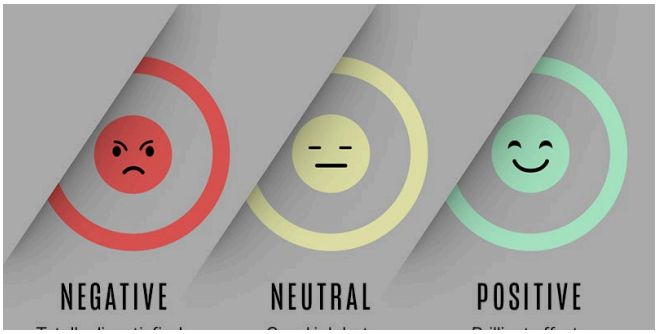
Data Scientist| NLP| DL| linkedin: https://www.linkedin.com/in/tanish-sharma-367102144/

No responses yet

Write a response

What are your thoughts?

More from TANISH SHARMA



TANISH SHARMA

Sentiment Analysis Using Pre-trained models and Transformer

Sentiment analysis is one of the most vital tasks in natural language processing, thus it ...

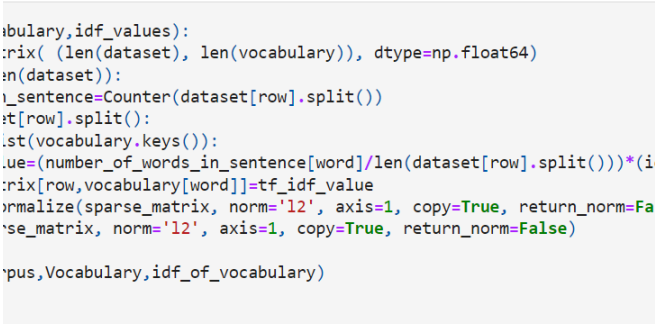
Jan 25, 2023 95 2

TANISH SHARMA

Fashion Product Recommendation System Using Resnet 50

Fashion is an ever-evolving industry that requires constant adaptation and innovation...

Mar 7, 2023 22



```
abulary,idf_values):
rix( (len(dataset), len(vocabulary)), dtype=np.float64)
n(dataset)):
_sentence=Counter(dataset[row].split())
t[row].split():
st(vocabulary.keys()):
ue=(number_of_words_in_sentence[word]/len(dataset[row].split()))*(i
rix[row,vocabulary[word]]=tf_idf_value
rmalize(sparse_matrix, norm='l2', axis=1, copy=True, return_norm=Fa
se_matrix, norm='l2', axis=1, copy=True, return_norm=False)

pus,Vocabulary,idf_of_vocabulary)
```

TANISH SHARMA

Sound Detection System(SDS)

Sound or voice detection has become a popular and important task in the audio sign...

Apr 10, 2023 2

TANISH SHARMA

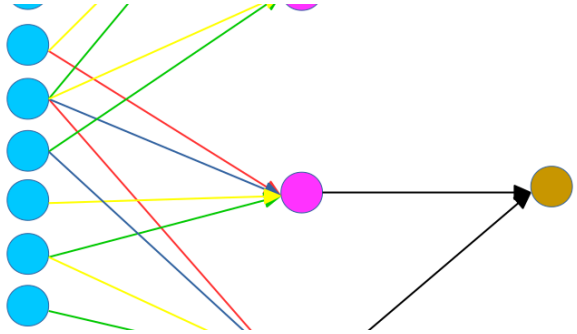
NLP

Implementation of Tf-Idf without using inbuilt sk-learn libraries.

Jun 21, 2022 35

See all from TANISH SHARMA

Recommended from Medium



Jo Wang

In The Deep Hub by Palash Mishra

CNN Basics: Convolutional Layers and Pooling Layer | How to...

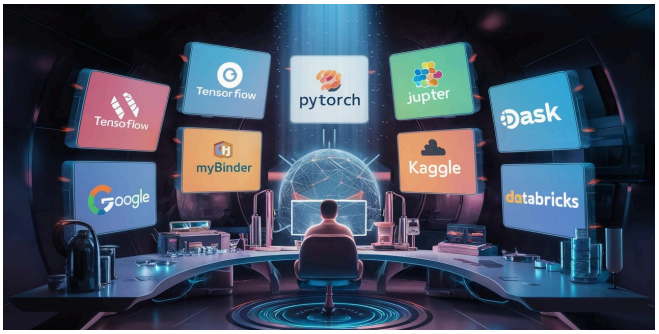
Key Ingredient 1: Convolutional Layers

Oct 29, 2024 31

Getting Started with PyTorch: A Beginner-Friendly Guide

If you've ever wondered how to build and train deep learning models, PyTorch is one of the...

Dec 3, 2024 55



In PythonForAll by Ahmad Waleed

Top 7 Free Alternatives to Google Colab for Data Science and...

Google Colab has been a go-to platform for many data scientists and machine learning...

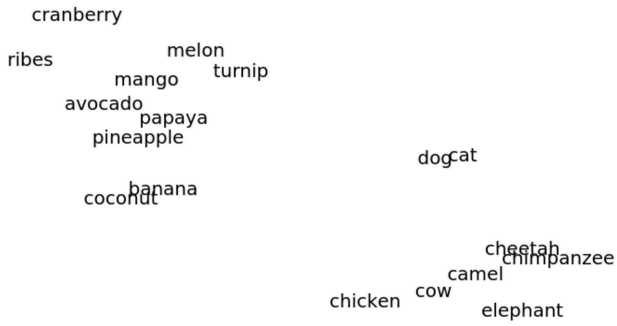
Oct 25, 2024 7

Papers in 100 Lines of Code

How to Train Image Translation Models Without Pairs of Images?

Discover the magic of CycleGAN and learn how to perform image translation without...

Nov 25, 2024 6



In AI-ML Interview Playbook by Sajid Khan

Q&A Crash Course On Computer Vision-Image Gen(Stable Diffusio...

This article is from my preparation strategy for an international client.

Santosh Pandey

How LLM Models Predict the Next Word: A Deep Dive into ChatGPT

Large Language Models (LLMs) like ChatGPT are trained to predict the next word in a...

Feb 4 5 Oct 22, 2024 14

See more recommendations