All You Need to Know About Skip Connections

Sivaram T — Published On August 24, 2021

Advanced Computer Vision Deep Learning Libraries Python PyTorch

Introduction

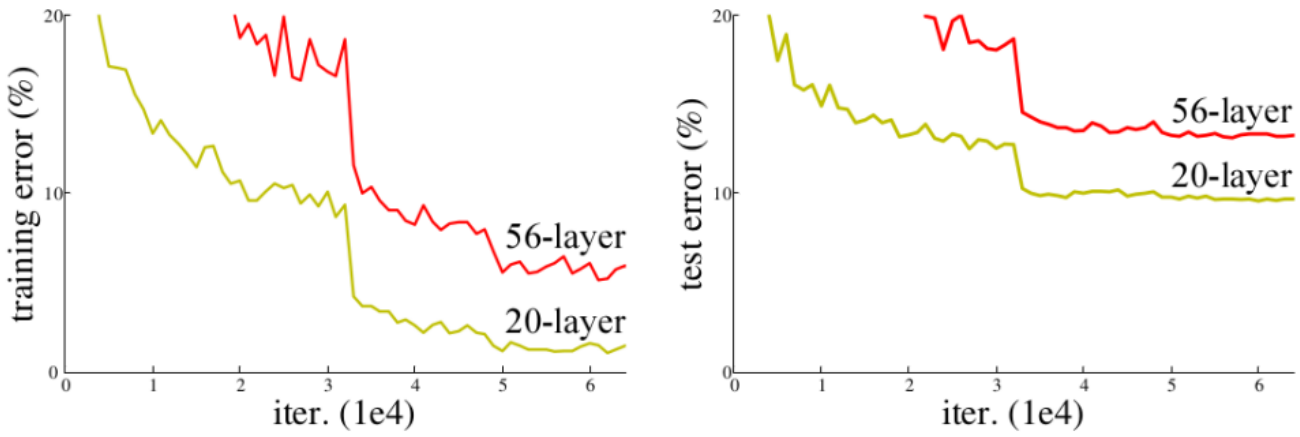
We need to train deeper networks to perform complex tasks. Training a deep neural net has a lot of complications not only limited to overfitting, high computation costs but also have some non-trivial problems. We're going to address these kinds of problems and how people in the Deep Learning community have solved these. Let's jump into the article!

Table of Contents

Why Skip Connections?

The beauty of *deep* neural networks is that they can learn complex functions more efficiently than their shallow counterparts. While training deep neural nets, the performance of the model drops down with the increase in depth of the architecture. This is known as the **degradation problem**. But, what could be the reasons for the saturation inaccuracy with the increase in network depth? Let us try to understand the reasons behind the degradation problem.

One of the possible reasons could be overfitting. The model tends to overfit with the increase in depth but that's not the case here. As you can infer from the below figure, the deeper network with 56 layers has more training error than the shallow one with 20 layers. **The deeper model doesn't perform as well as the shallow one.** Clearly, overfitting is not the problem here.



Train and test error for 20-layer and 56-layer NN

Another possible reason can be vanishing gradient and/or exploding gradient problems. However, the authors of ResNet (He et al.) argued that the use of Batch Normalization and proper initialization of weights through normalization ensures that the gradients have healthy norms. But, what went wrong here? Let's understand this by construction.

Consider a shallow neural network that was trained on a dataset. Also, consider a deeper one in which the initial layers have the same weight matrices as the shallow network (the blue colored layers in the below diagram) with added some extra layers (green colored layers). We set the weight matrices of the added layers as *identity matrices* (identity mappings).

All You Need to Know About Skip Connections

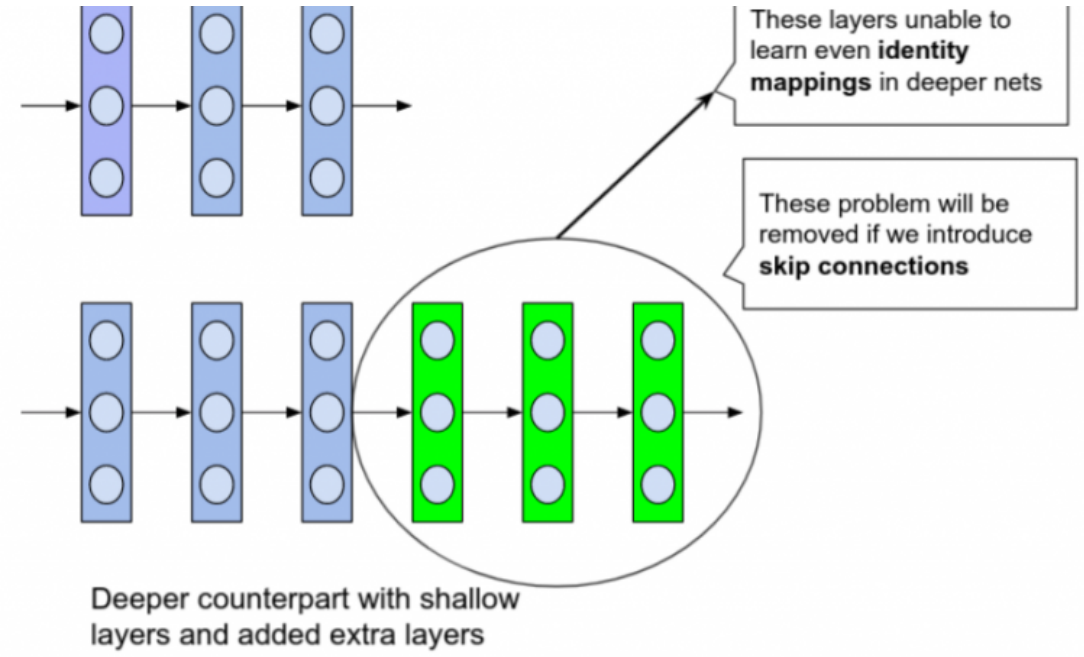


Diagram explaining the construction

From this construction, the deeper network should not produce any higher training error than its shallow counterpart because we are actually using the shallow model's weight in the deeper network with added identity layers. But experiments prove that the deeper network produces high training error comparing to the shallow one. This states the **inability of deeper layers to learn even identity mappings**.

The degradation of training accuracy indicates that *not all systems are similarly easy to optimize.*

One of the primary reasons is due to random initialization of weights with a mean around zero, L1, and L2 regularization. As a result, the weights in the model would always be around zero and thus the deeper layers can't learn identity mappings as well.

Here comes the concept of **skip connections** which would enable us to train very deep neural networks. Let's learn this awesome concept now.

What are Skip Connections?

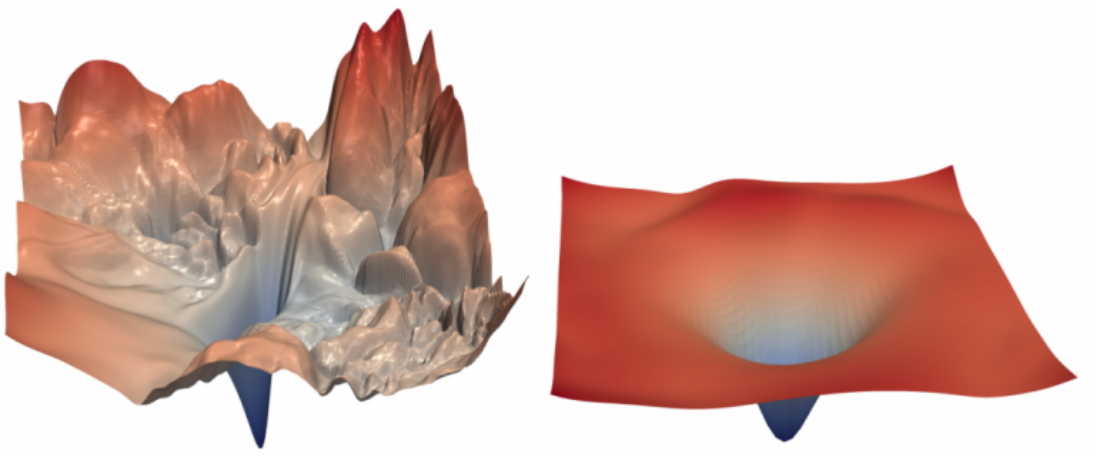
Skip Connections (or Shortcut Connections) as the name suggests *skips some of the layers in the neural network and feeds the output of one layer as the input to the next layers*.

Skip Connections were introduced to solve different problems in different architectures. In the case of ResNets, skip connections solved the *degradation problem* that we addressed earlier whereas, in the case of DenseNets, it ensured **feature reusability**. We'll discuss them in detail in the following sections.

Skip connections were introduced in literature even before residual networks. For example, **Highway Networks** (Srivastava et al.) had skip connections with *gates* that controlled and learned the flow of information to deeper layers. This concept is similar to the gating mechanism in LSTM. Although ResNets is actually a special case of Highway networks, the performance isn't up to the mark comparing to ResNets. This suggests that it's better to keep the gradient highways *clear* than to go for any gates – simplicity wins here!

Neural networks can learn any functions of arbitrary complexity, which could be high-dimensional and non-convex. Visualizations have the potential to help us answer several important questions about why neural networks work. And there is actually some nice work done by Li et al. which enables us to visualize the complex loss surfaces. The results from the networks

All You Need to Know About Skip Connections



(a) without skip connections (b) with skip connections

The loss surfaces of ResNet-56 with and without skip connections

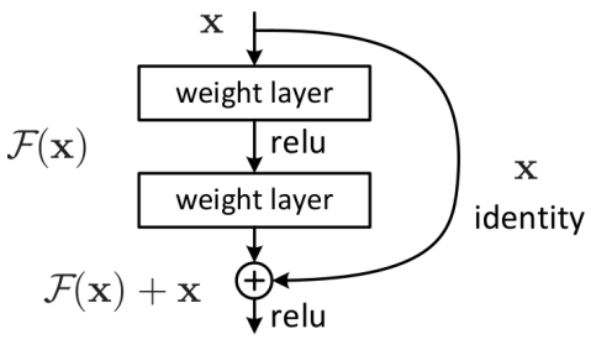
As you can see here, the loss surface of the neural network with skip connections is smoother and thus leading to faster convergence than the network without any skip connections. Let's see the variants of skip connections in the next section.

Variants of Skip Connections

In this section, we will see the variants of skip connections in different architectures. Skip Connections can be used in 2 fundamental ways in Neural Networks: Addition and Concatenation.

Residual Networks (ResNets)

Residual Networks were proposed by He et al. in 2015 to solve the image classification problem. In ResNets, the information from the initial layers is passed to deeper layers by matrix addition. This operation doesn't have any additional parameters as the output from the previous layer is added to the layer ahead. A single residual block with skip connection looks like this:



A residual block

Thanks to the deeper layer representation of ResNets as pre-trained weights from this network can be used to solve multiple tasks. It's not only limited to image classification but also can solve a wide range of problems on image segmentation, keypoint detection & object detection. Hence, ResNet is one of the most influential architectures in the deep learning community.

Next, we'll learn about another variant of skip connections in DenseNets which is inspired by ResNets.

I would recommend you to go through the below resources for an in-detailed understanding of ResNets –

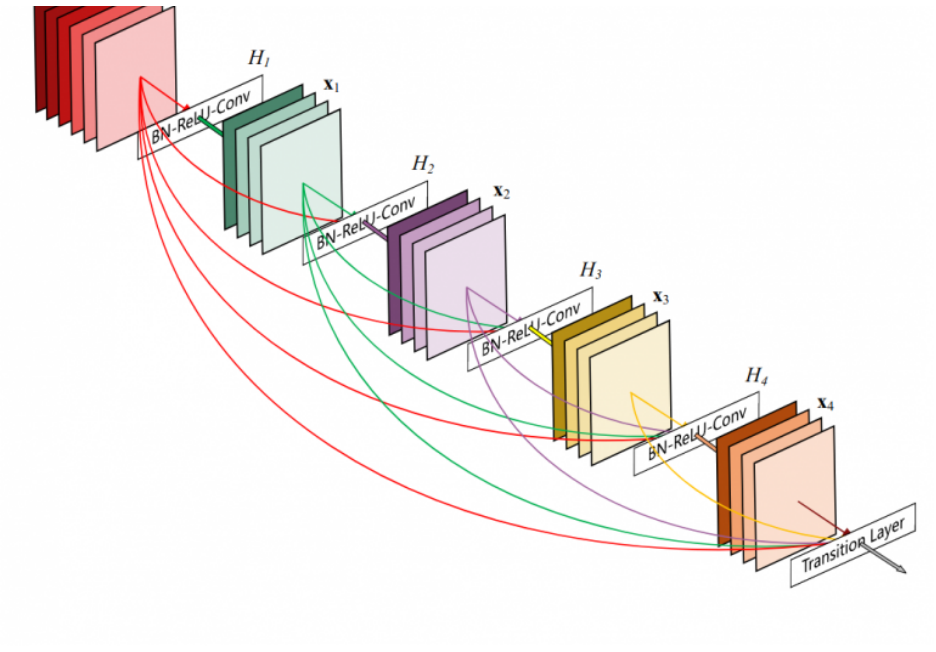
- Understanding ResNet and analyzing various models on the CIFAR-10 dataset

Densely Connected Convolutional Networks (DenseNets)

DenseNets were proposed by Huang et al. in 2017. The primary difference between ResNets and DenseNets is that DenseNets concatenates the output feature maps of the layer with the next layer rather than a summation.

Coming to Skip Connections, DenseNets uses Concatenation whereas ResNets uses Summation

All You Need to Know About Skip Connections

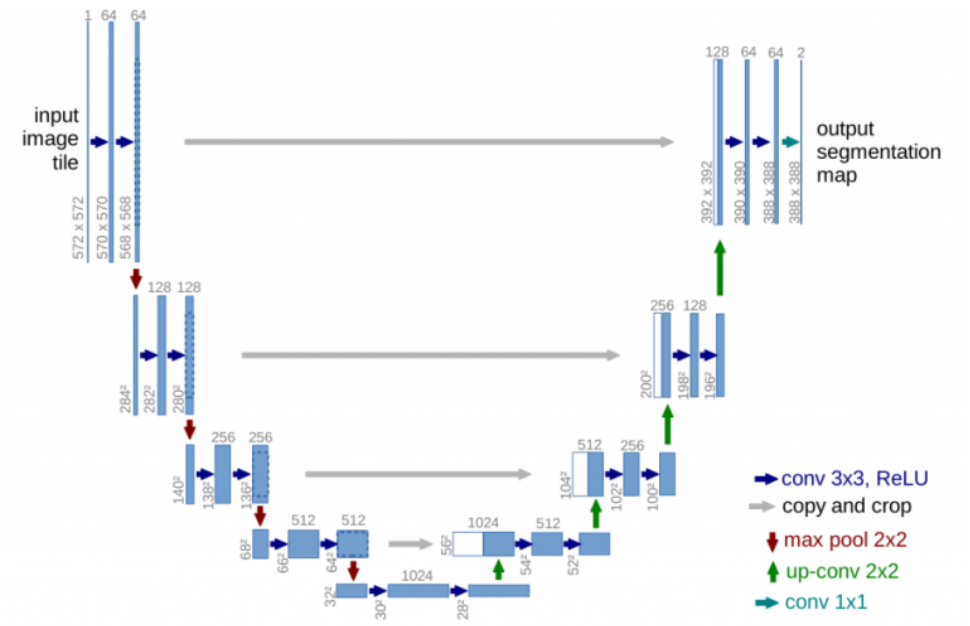


A 5-layer dense block

The idea behind the concatenation is to use features that are learned from earlier layers in deeper layers as well. This concept is known as **Feature Reusability**. So, DenseNets can learn mapping with fewer parameters than a traditional CNN as there is no need to learn redundant maps.

U-Net: Convolutional Networks for Biomedical Image Segmentation

The use of skip connections influences the field of biomedical too. U-Nets were proposed by Ronneberger et al. for biomedical image segmentation. It has an **encoder-decoder** part including S*kip Connections.* The overall architecture looks like the English letter "U", thus the name U-Nets.



U-Net architecture

The layers in the encoder part are skip connected and concatenated with layers in the decoder part (those are mentioned as grey lines in the above diagram). This makes the U-Nets use *fine-grained* details learned in the encoder part to construct an image in the decoder part.

These kinds of connections are *long skip connections* whereas the ones we saw in ResNets were *short skip connections*. More about U-Nets here.

Okay! Enough of theory, let's implement a block of the discussed architectures and how to load and use them in PyTorch!

Implementation of Skip Connections

In this section, we will build ResNets and DesNets using Skip Connections from the scratch. Are you excited? Let's go!

ResNet – A residual block

All You Need to Know About Skip Connections

oriented structure.

```python
# import required libraries
import torch
from torch import nn
import torch.nn.functional as F
import torchvision

# basic resdidual block of ResNet
# This is generic in the sense, it could be used for downsampling of features.
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=[1, 1], downsample=None):
        """
        A basic residual block of ResNet
        Parameters
        ----------
            in_channels: Number of channels that the input have
            out_channels: Number of channels that the output have
            stride: strides in convolutional layers
            downsample: A callable to be applied before addition of residual mapping
        """
        super(ResidualBlock, self).__init__()

        self.conv1 = nn.Conv2d(
            in_channels, out_channels, kernel_size=3, stride=stride[0],
            padding=1, bias=False
        )

        self.conv2 = nn.Conv2d(
            out_channels, out_channels, kernel_size=3, stride=stride[1],
            padding=1, bias=False
        )

        self.bn = nn.BatchNorm2d(out_channels)
        self.downsample = downsample

    def forward(self, x):
        residual = x
        # applying a downsample function before adding it to the output
        if(self.downsample is not None):
            residual = downsample(residual)

        out = F.relu(self.bn(self.conv1(x)))

        out = self.bn(self.conv2(out))
        # note that adding residual before activation
        out = out + residual
        out = F.relu(out)
        return out
```

residual_block.py hosted with ❤ by GitHub view raw

As we have a Residual block in our hand, we can build a ResNet model of arbitrary depth! Let's quickly build the first five layers of **ResNet-34** to get an idea of how to connect the residual blocks.

```python
# downsample using 1 * 1 convolution
downsample = nn.Sequential(
    nn.Conv2d(64, 128, kernel_size=1, stride=2, bias=False),
    nn.BatchNorm2d(128)
)
# First five layers of ResNet34
resnet_blocks = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False),
    nn.MaxPool2d(kernel_size=2, stride=2),
    ResidualBlock(64, 64),
```

All You Need to Know About Skip Connections

```
)

# checking the shape
inputs = torch.rand(1, 3, 100, 100) # single 100 * 100 color image
outputs = resnet_blocks(inputs)
print(outputs.shape)    # shape would be (1, 128, 13, 13)
```

main_resnet.py hosted with ❤ by GitHub view raw

PyTorch provides us an easy way to load ResNet models with pretrained weights trained on the ImageNet dataset.

```
# one could also use pretrained weights of ResNet trained on ImageNet
resnet34 = torchvision.models.resnet34(pretrained=True)
```

resnet_pretrained.py hosted with ❤ by GitHub view raw

DenseNet – A dense block

Implementing the complete densenet would be a little bit complex. Let's grab it step by step.

1. Implement a DenseNet layer
2. Build a dense block
3. Connect multiple dense blocks to obtain a densenet model

```
class Dense_Layer(nn.Module):
    def __init__(self, in_channels, growthrate, bn_size):
        super(Dense_Layer, self).__init__()

        self.bn1 = nn.BatchNorm2d(in_channels)
        self.conv1 = nn.Conv2d(
            in_channels, bn_size * growthrate, kernel_size=1, bias=False
        )
        self.bn2 = nn.BatchNorm2d(bn_size * growthrate)
        self.conv2 = nn.Conv2d(
            bn_size * growthrate, growthrate, kernel_size=3, padding=1, bias=False
        )

    def forward(self, prev_features):
        out1 = torch.cat(prev_features, dim=1)
        out1 = self.conv1(F.relu(self.bn1(out1)))
        out2 = self.conv2(F.relu(self.bn2(out1)))
        return out2
```

dense_layer.py hosted with ❤ by GitHub view raw

Next, we'll implement a dense block that consists of an arbitrary number of DenseNet layers.

```
class Dense_Block(nn.ModuleDict):
    def __init__(self, n_layers, in_channels, growthrate, bn_size):
        """
        A Dense block consists of `n_layers` of `Dense_Layer`
        Parameters
        ----------
            n_layers: Number of dense layers to be stacked
            in_channels: Number of input channels for first layer in the block
            growthrate: Growth rate (k) as mentioned in DenseNet paper
            bn_size: Multiplicative factor for # of bottleneck layers
        """
        super(Dense_Block, self).__init__()

        layers = dict()
        for i in range(n_layers):
            layer = Dense_Layer(in_channels + i * growthrate, growthrate, bn_size)
            layers['dense{}'.format(i)] = layer

```

All You Need to Know About Skip Connections

```
    def forward(self, features):
        if(isinstance(features, torch.Tensor)):
            features = [features]

        for _, layer in self.block.items():
            new_features = layer(features)
            features.append(new_features)

        return torch.cat(features, dim=1)
```

dense_block.py hosted with ❤ by GitHub view raw

From the dense block, let's build DenseNet. Here, I've omitted the transition layers of DenseNet architecture (which acts as downsampling) for simplicity.

```
# a block consists of initial conv layers followed by 6 dense layers
dense_block = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=7, padding=3, stride=2, bias=False),
    nn.BatchNorm2d(64),
    nn.MaxPool2d(3, 2),
    Dense_Block(6, 64, growthrate=32, bn_size=4),
)

inputs = torch.rand(1, 3, 100, 100)
outputs = dense_block(inputs)
print(outputs.shape)    # shape would be (1, 256, 24, 24)

# one could also use pretrained weights of DenseNet trained on ImageNet
densenet121 = torchvision.models.densenet121(pretrained=True)
```

main_densenet.py hosted with ❤ by GitHub view raw

End Notes

In this article, we've discussed the importance of skip connections for the training of deep neural nets and how skip connections were used in ResNet, DenseNet, and U-Net with its implementation. I know, this article covers many theoretical aspects which are not easy to grasp in one go. So, feel free to leave comments if you have any.

Have a nice day!

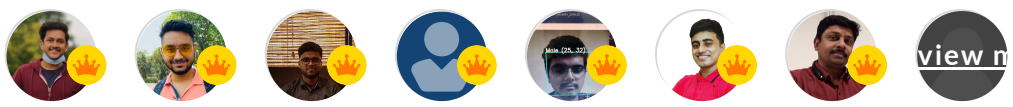
Computer Vision deep learning deep learning architectures python Theory

All You Need to Know About Skip Connections

About the Author

Sivaram T

Our Top Authors



Download

Analytics Vidhya App for the Latest blog/Article

Previous Post

Everything a Beginner should know about Classes and Objects in Python Before Starting Your Data Science Journey[With Examples]

Next Post

NLPAUG – A Python library to Augment Your Text Data

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

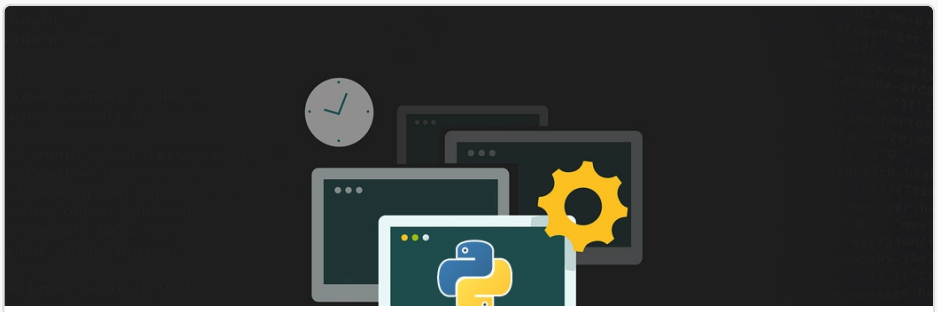
Website

Notify me of follow-up comments by email.

All You Need to Know About Skip Connections

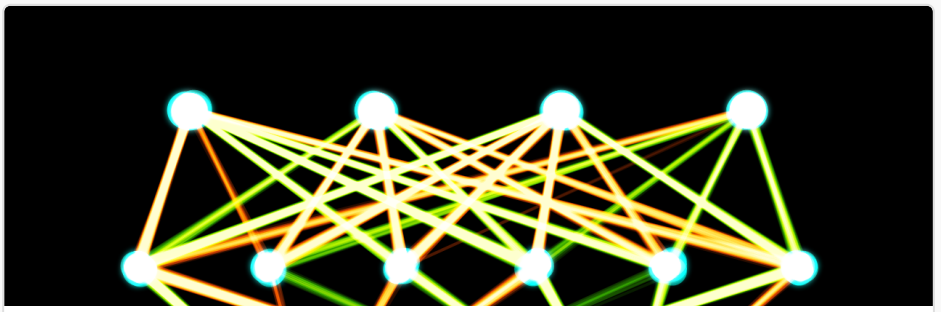
Submit

Top Resources



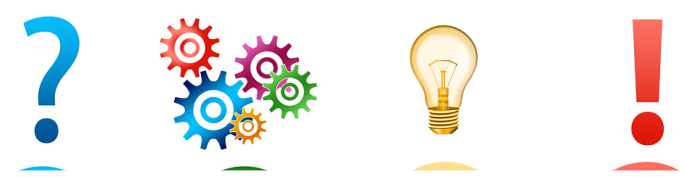
Python Tutorial: Working with CSV file for Data Science

Harika Bonthu - AUG 21, 2021



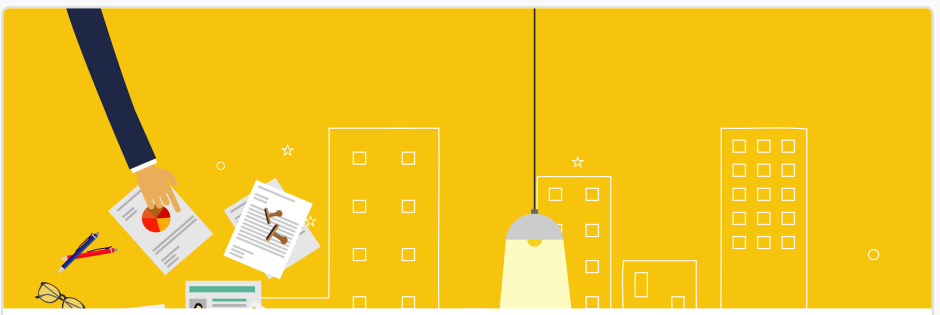
Boost Model Accuracy of Imbalanced COVID-19 Mortality Prediction Using GAN-based..

Bala Gangadhar Thilak Adiboina - OCT 07, 2020



40 Questions to test a data scientist on Machine Learning..

1201904 - APR 30, 2017



Python Coding Interview Questions for Freshers

Saumyab271 - JUL 23, 2022