# An Overview On U-net Architecture

Arun Mohan · Follow
Published in DataDrivenInvestor
9 min read · Nov 26, 2019

Listen Share More

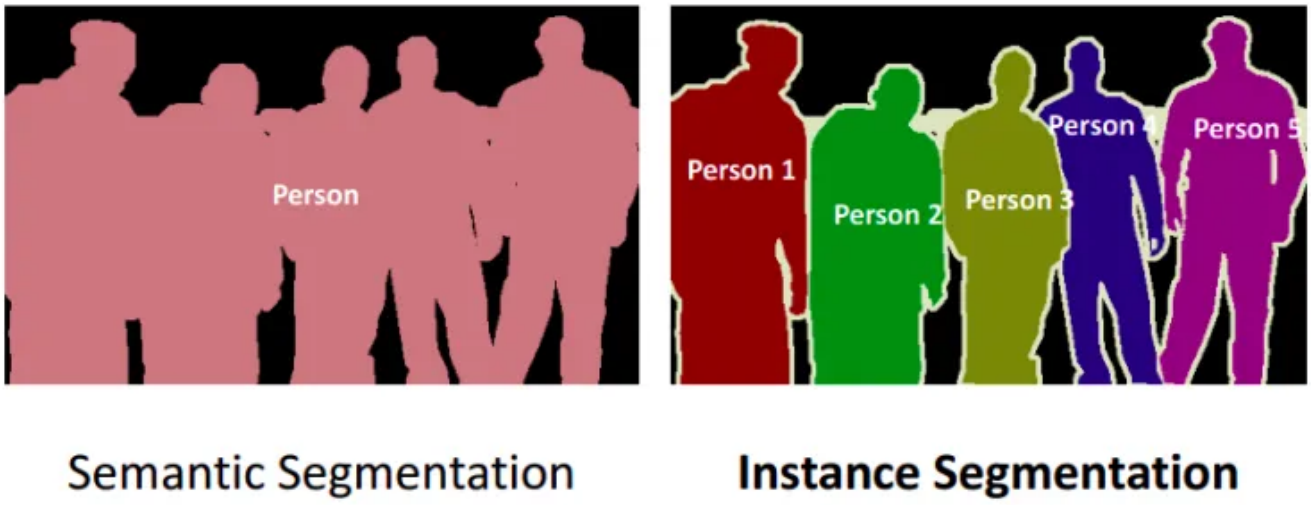
U-Net is considered as one of the standard CNN architectures for image classification tasks. It is considered as a best network for fast and precise segmentation of images.Before getting deep into U-net we will explain about Some concepts related to it.

### Semantic segmentation

Goal of semantic segmentation is to label each pixel of image with corresponding class of what is being represented. That is it refers to the process of linking each pixel in an image to a class label. These labels could include a person, car, flower, piece of furniture, etc., just to mention a few.

### Instance segmentation

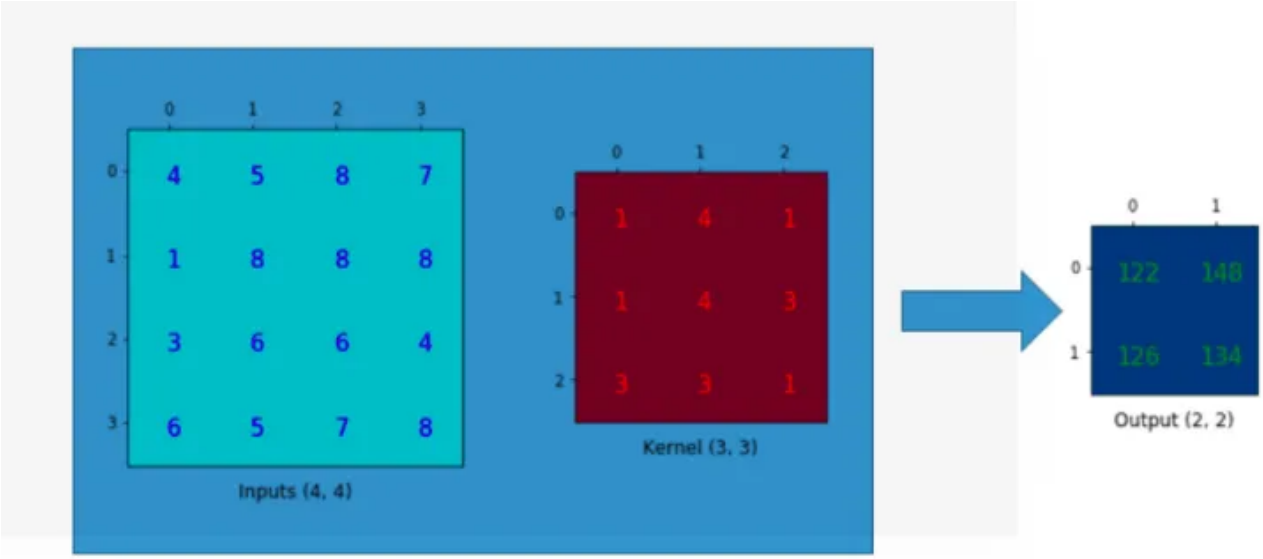
Along with pixel level classification, we expect the computer to classify each instance of class separately. It is called instance segmentation.That is different instances of the same class are segmented individually in instance segmentation.




Difference in labels in semantic and instance segmentation

**Convolution**

Let us explain what is convolution and how it can be represented as matrix in simple terms.




Convolution example

Here we used a (3 x 3) kernel with an (4 x 4) input to generate a (2 x 2) output. Here we have zero padding and we are using striding as 1. The output of convolution entirely depends on kernel size, padding ,striding and no of input.

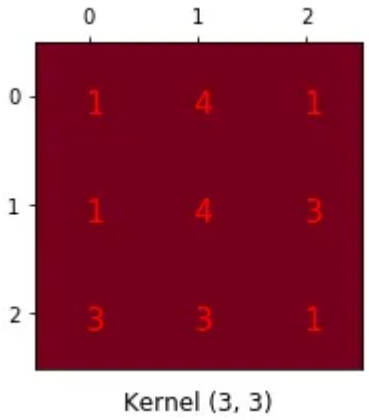
**Cognitive computing - a skill-set widely considered to be the most vital manifestation of...**

Here the convolution operation do the element-wise multiplication and find its sum between the input matrix and kernel matrix. Since we have no padding and the stride of 1, we can do this only 4 times. Hence, the output matrix is 2 x 2 .

Now we will explain it using a convolution matrix.

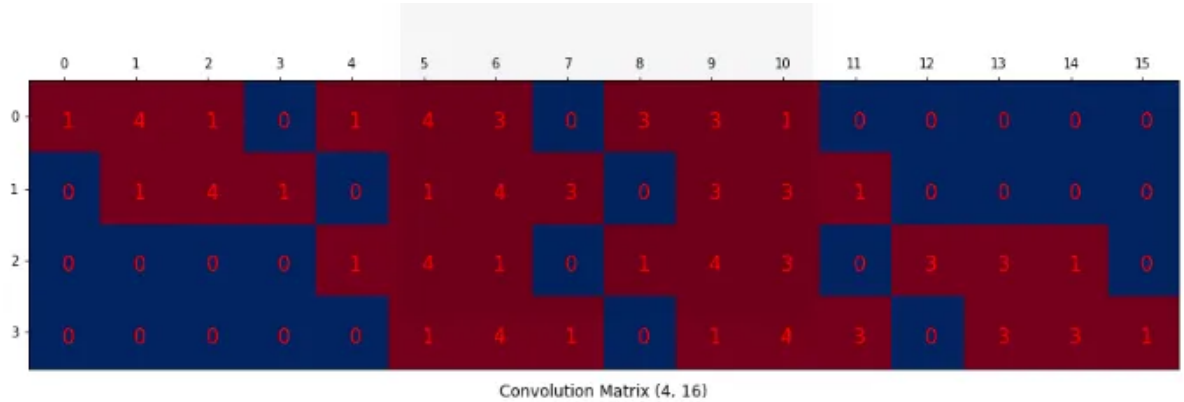
Step 1:

Consider we are using a 3 x 3 kernel as below:




Kernel (3, 3)

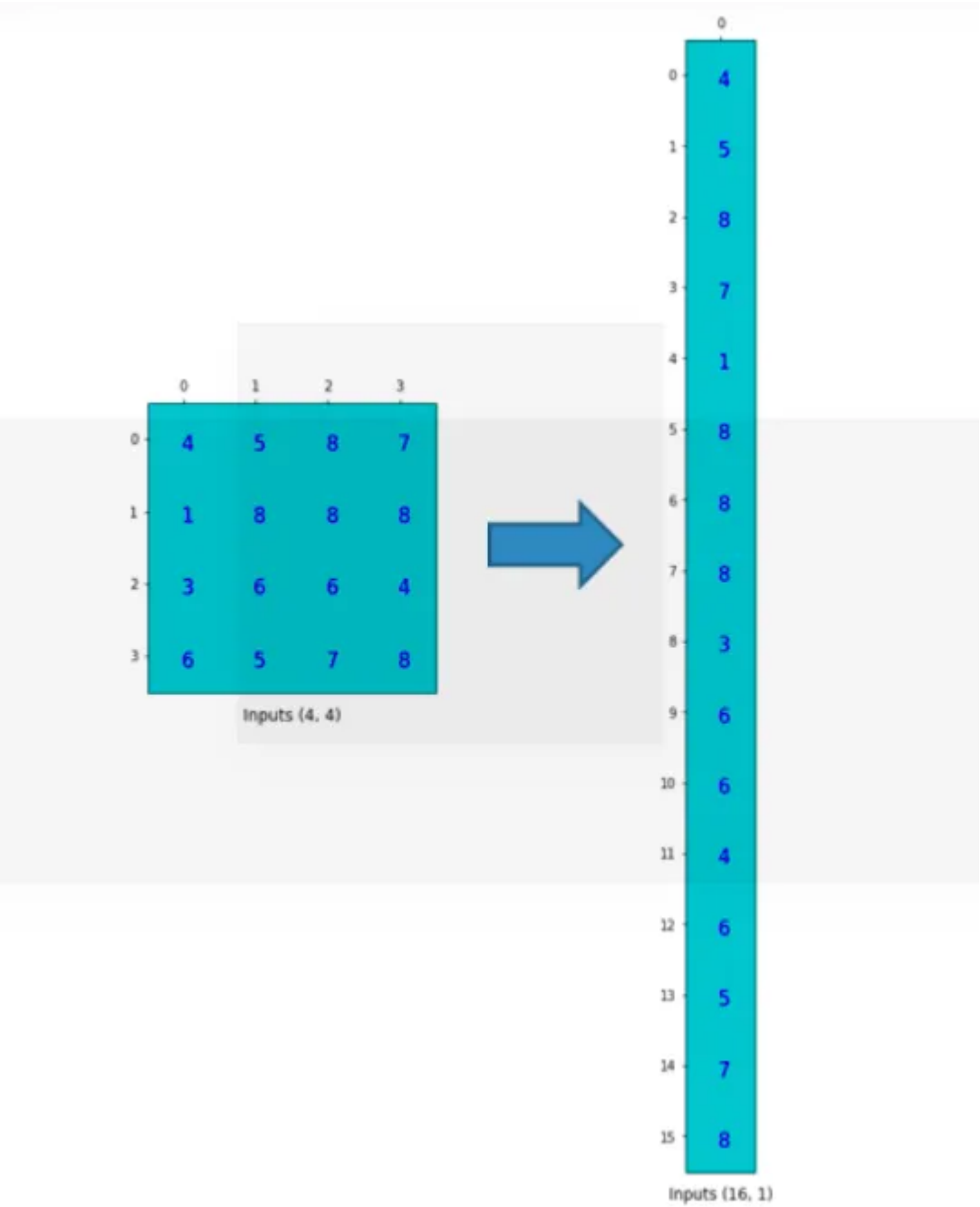
We arrange 3 x 3 kernel to 4 x 16 convolution matrix as given below:




Convolution Matrix (4, 16)

Step 2:

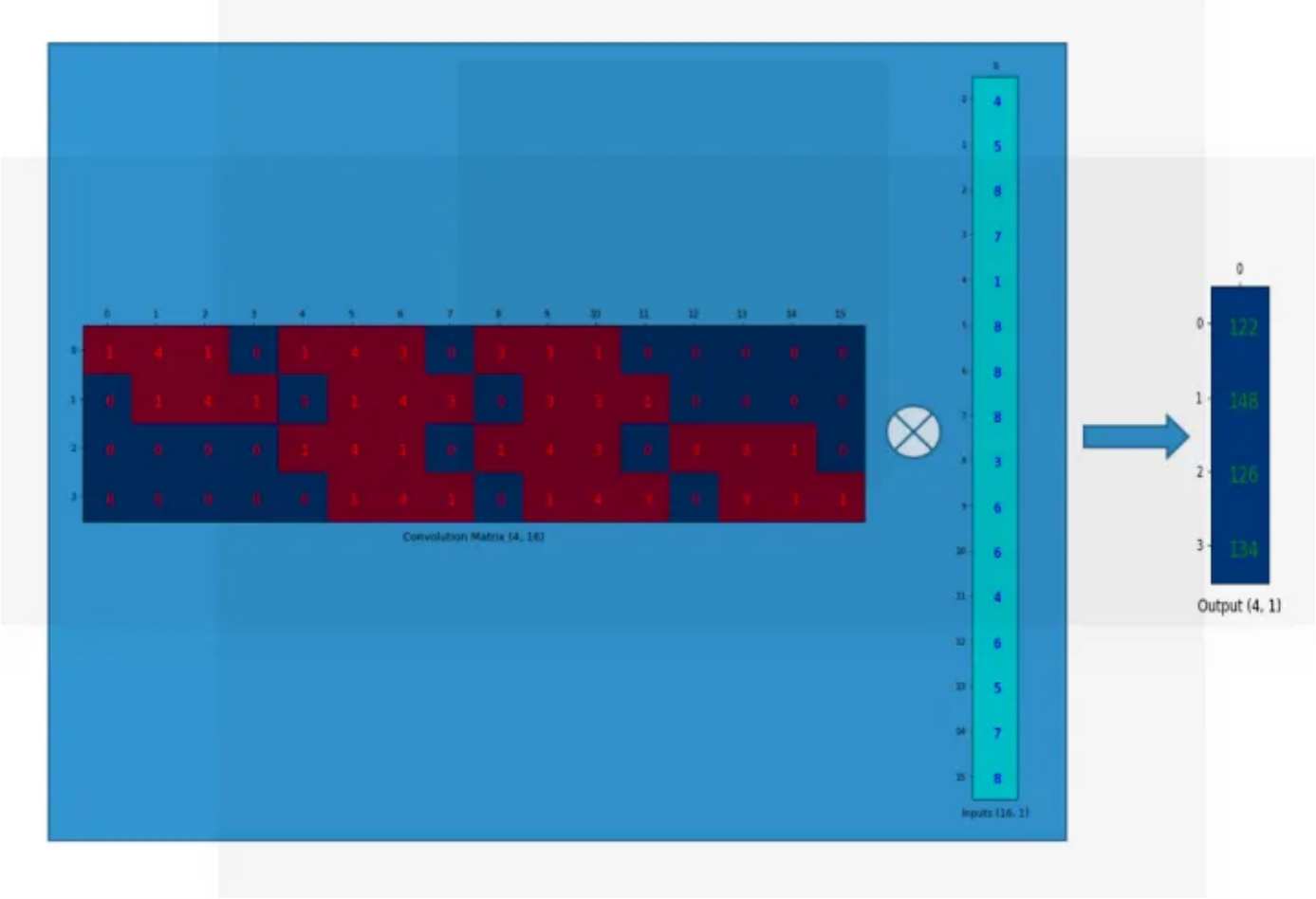
Represent our input matrix as a column vector as given below:

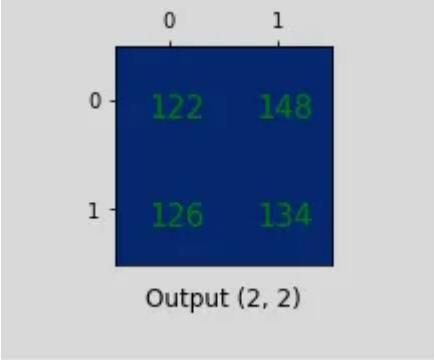



4 × 4 input matrix represented as 16 × 1 column matrix.

Step 3:

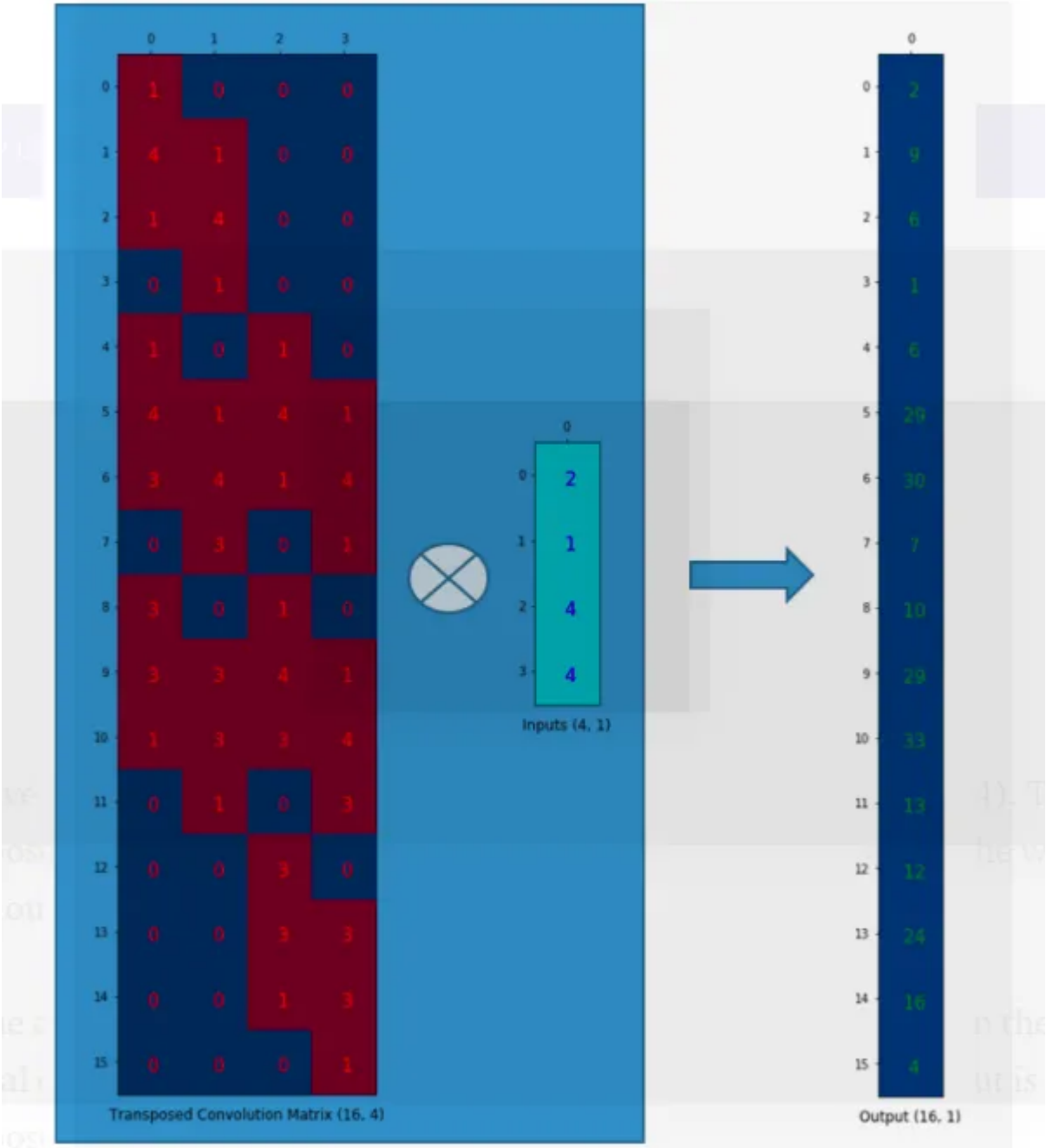
As final step we will multiply our 16 x 4 kernel matrix with flattened input matrix to get flattened type of output matrix. We rearrange it to give real output matrix as follows.



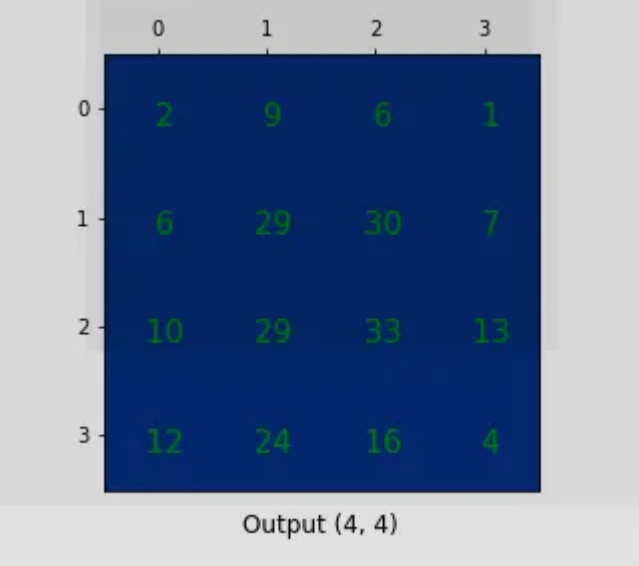




**Transposed Convolution**

We want to go from 2 x 2 matrix to 4 x 4 matrix. In this case we can use transposed convolution. It is similar to convolution matrix operation we did earlier. Only difference is that here we multiply transposed kernel matrix with flattened input. Transposed convolution (sometimes also called as de convolution or fractionally strided convolution) is a technique to perform up sampling of an image with learnable parameters.

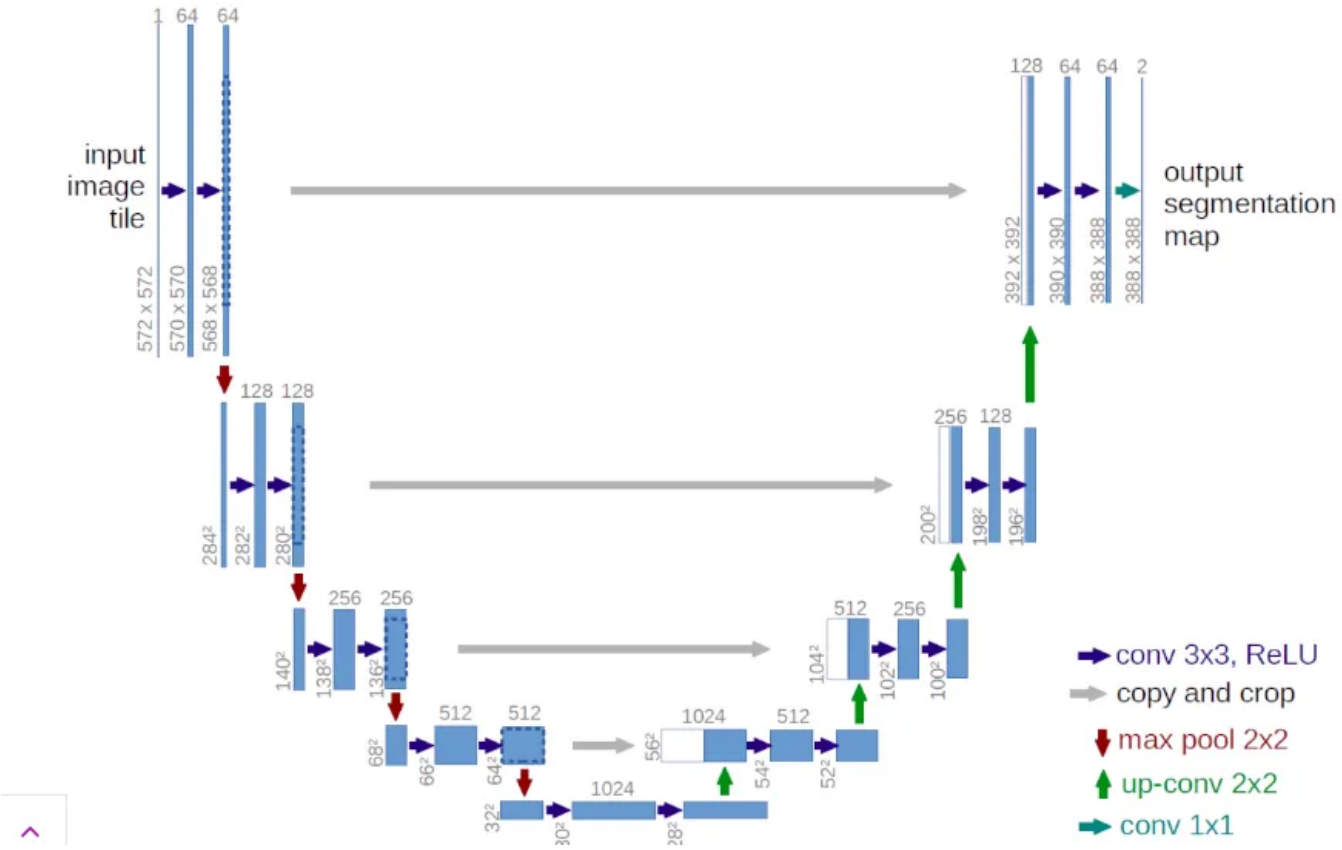



The output can be reshaped into 4 x 4 matrix.




We have just up-sampled a smaller matrix (2 x 2) into a larger one (4 x 4). In simple words we can say that a transposed convolution is exactly the opposite process of a normal convolution i.e., the input volume is a low resolution image and the output volume is a high resolution image.

## U-net Architecture



The *UNET is* an architecture which was developed by Olaf Ronneberger et al. for BioMedical Image Segmentation. It mainly consists of two paths. One is an ecoder path and other is a decoder path. The encoder path captures the context of the image producing feature maps. Encoder path is just a stack of convolution and max pooling layers. Decoder path used to enable precise localization using transposed convolutions. U-net only contains Convolutional layers and does not contain any Dense layer because of which it can accept image of any size.

**Contraction/down sampling path(Encoder Path):**

- The encoding path is composed of 4 blocks.
- Each block consist of

1. Two 3 x 3 convolution layers + ReLU activation function (with batch normalization)
2. One 2 x 2 max pooling layer.

- Note that in the original paper, the size of the input image is 572 x 572 x 3. Using 64 (3 x 3) kernels, convolution is done and produce a feature map of 570 x 570 x 64. Now it is again multiplied with 64(3 x 3) kernels and produce feature map of 568 x 568 x 64. Now a max pooling is done using 2 x 2 kernel to produce feature map of 284 x 284 x 64.
- We have to note that the number of feature maps doubles at each pooling, starting with 64 feature maps for the first block, 128 for the second, and so on. The purpose of this contracting path is to capture the context of the input image in order to be able to do segmentation.

**Expansion/Up sampling path(Decoder Path)**

- The expansion path consists of 4 blocks. Each block consists of:

1. Deconvolution layer with stride 2.
2. Concatenation with the corresponding cropped feature map from the contracting path. ie To get better precise locations, at every step of the decoder we use skip connections by concatenating the output of the transposed convolution layers with the feature maps from the Encoder at the same level.
3. Two 3 x 3 convolution layers + ReLU activation function (with batch normalization).

**Advantages**

- The U-Net combines the location information from the downsampling path with the contextual information in the upsampling path to finally obtain a general information combining localisation and context, which is necessary to predict a good segmentation map.
- No dense layer, so images of different sizes can be used as input (since the only parameters to learn on convolution layers are the kernel, and the size of the kernel is independent from input image size).
- The use of massive data augmentation is important in domains like biomedical segmentation, since the number of annotated samples is usually limited.

## Detail implementation of U-net using Keras

Below shows the detailed implementation of unet in keras:

```
def get_unet(IMG_WIDTH=256,IMG_HEIGHT=256,IMG_CHANNELS=3):
    inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
    s = Lambda(lambda x: x / 255) (inputs)

    c1 = Conv2D(16, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (s)
    c1 = Dropout(0.1) (c1)
    c1 = Conv2D(16, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c1)
    p1 = MaxPooling2D((2, 2)) (c1)

    c2 = Conv2D(32, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (p1)
    c2 = Dropout(0.1) (c2)
    c2 = Conv2D(32, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c2)
    p2 = MaxPooling2D((2, 2)) (c2)

    c3 = Conv2D(64, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (p2)
    c3 = Dropout(0.2) (c3)
    c3 = Conv2D(64, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c3)
    p3 = MaxPooling2D((2, 2)) (c3)

    c4 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (p3)
    c4 = Dropout(0.2) (c4)
    c4 = Conv2D(128, (3, 3), activation='elu',
```

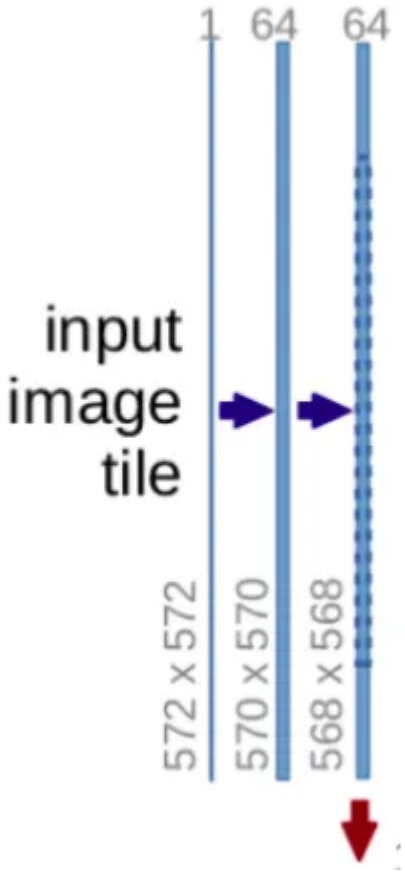
```
kernel_initializer='he_normal', padding='same') (c4)
    p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

    c5 = Conv2D(256, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (p4)
    c5 = Dropout(0.3) (c5)
    c5 = Conv2D(256, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c5)

    u6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')
(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same')(u6)
    c6 = Dropout(0.2) (c6)
    c6 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c6)

u7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (c6)
    u7 = concatenate([u7, c3])
    c7 = Conv2D(64, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (u7)
    c7 = Dropout(0.2) (c7)
    c7 = Conv2D(64, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c7)

u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c7)
    u8 = concatenate([u8, c2])
    c8 = Conv2D(32, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (u8)
    c8 = Dropout(0.1) (c8)
    c8 = Conv2D(32, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c8)

u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = Conv2D(16, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (u9)
    c9 = Dropout(0.1) (c9)
    c9 = Conv2D(16, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c9)

outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)

model = Model(inputs=[inputs], outputs=[outputs])
    model.compile(optimizer='adam',loss='binary_crossentropy',
metrics=[dice_coef,mean_iou])
    return model
```

Now we will go through code part by part. The structure of the contraction path of U-net is as follows:

```
conv layer1 -> conv layer2 -> max pooling -> dropout(optional)
```

First part of our code is:

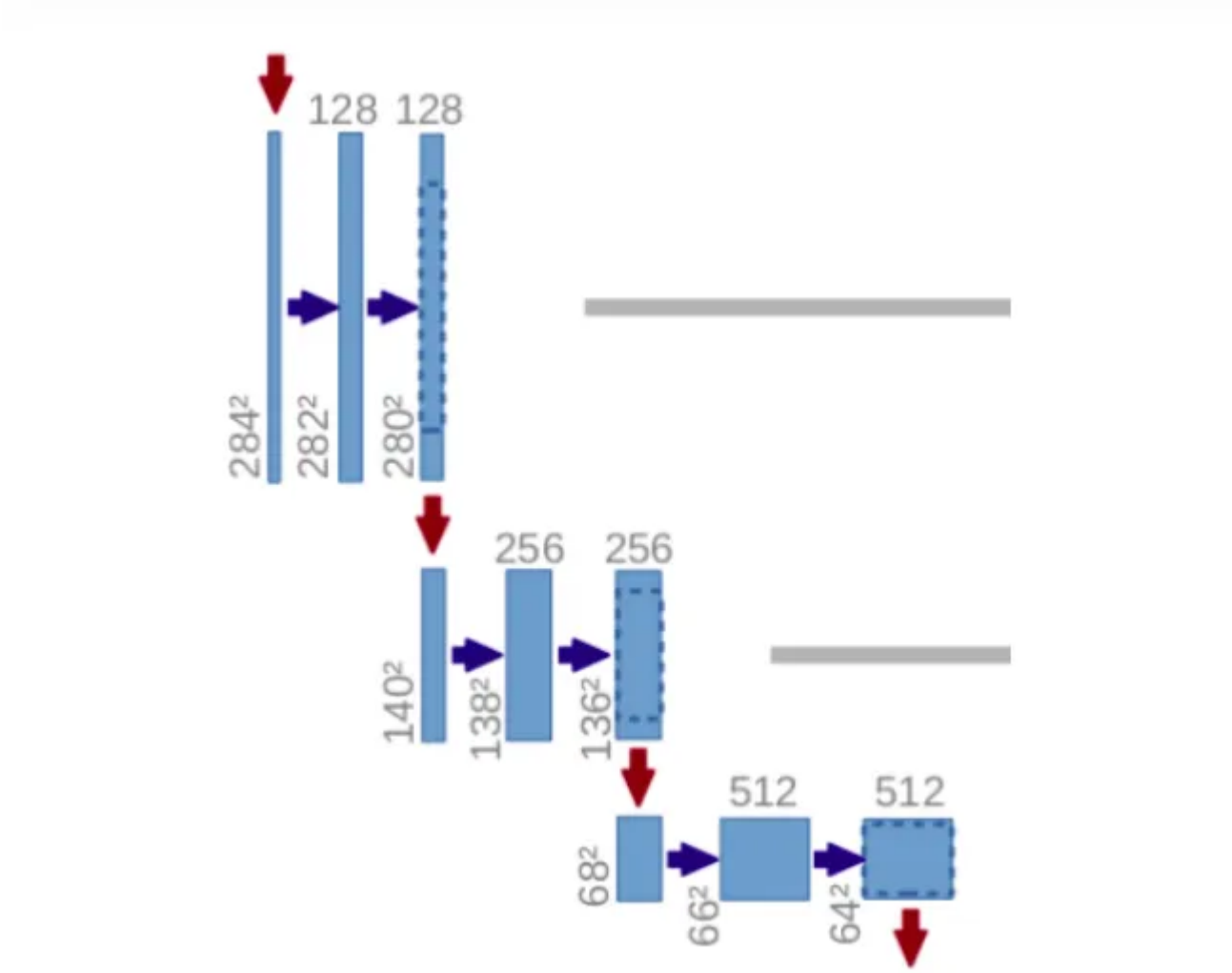
```
c1 = Conv2D(16,(3,3),activation='elu',
kernel_initializer='he_normal', padding='same') (s)
c1 = Dropout(0.1) (c1)
c1 = Conv2D(16, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c1)
p1 = MaxPooling2D((2, 2)) (c1)
```




Note that each block consist of three convolution layers.Here in research paper input is of 572 x 572 x 1 dimension and it goes through convolution twice with 64 channels. The number of channel will change from 1 → 64, as convolution process proceeds. The red

arrow in bottom indicates 2 x 2 max pooling layer which half down the size of the image(568 x 568 to 284 x 284).

These type of blocks are repeated three times as follows:

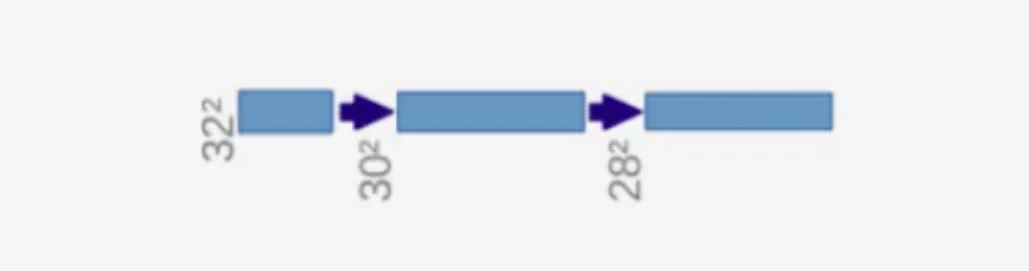



The code for same is:

```
c2 = Conv2D(32, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (p1)
c2 = Dropout(0.1) (c2)
c2 = Conv2D(32, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c2)
p2 = MaxPooling2D((2, 2)) (c2)

c3 = Conv2D(64, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (p2)
c3 = Dropout(0.2) (c3)
```

```
c3 = Conv2D(64, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c3)
p3 = MaxPooling2D((2, 2)) (c3)

c4 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (p3)
c4 = Dropout(0.2) (c4)
c4 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c4)
p4 = MaxPooling2D(pool_size=(2, 2)) (c4)
```

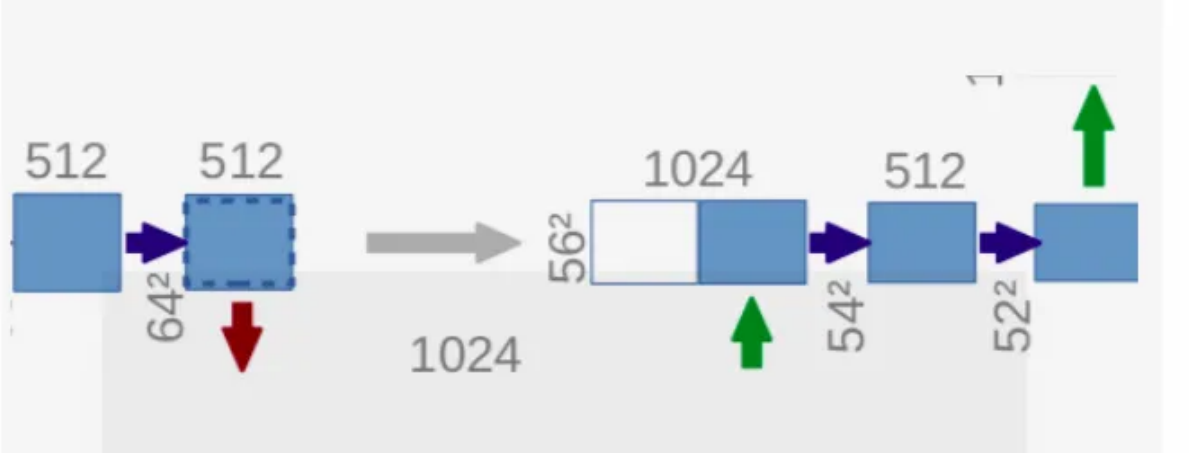
Now we reaches a bottom layer with no maxpooling. It also consist of two convolution layers.




```
c5 = Conv2D(256, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (p4)
c5 = Dropout(0.3) (c5)
c5 = Conv2D(256, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c5)
```

The image at this moment has been resized to 28 x 28 x 1024. Now we will go through expansive path.

The structure of the expansive path of U-net is as follows:




```
conv 2d_transpose -> concatenate -> conv layer1 -> conv layer2
```

First part of expansive path is:

```
u6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same') (c5)
u6 = concatenate([u6, c4])
c6 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same')(u6)
c6 = Dropout(0.2) (c6)
c6 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c6)
```

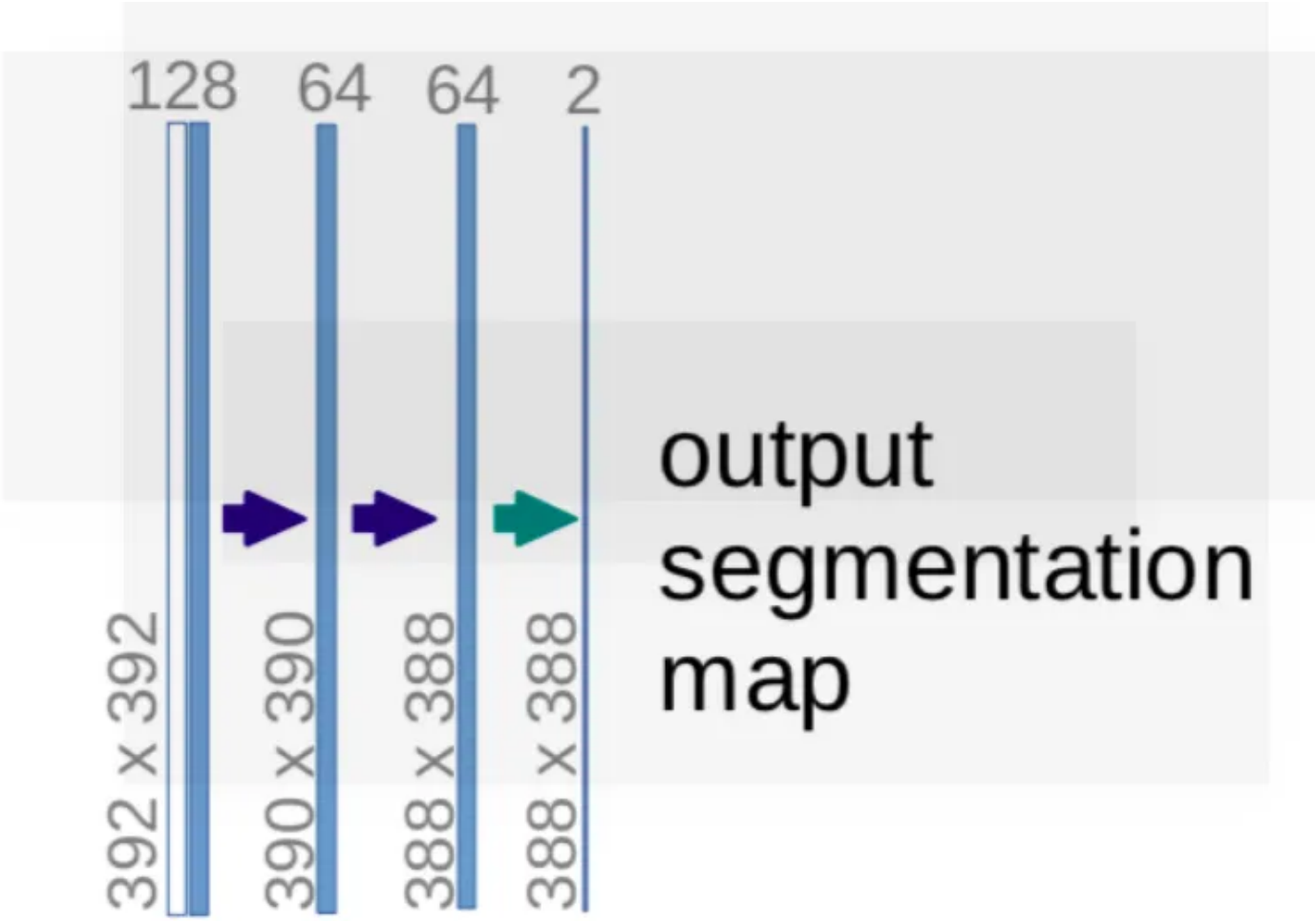
Transposed convolution as explained earlier is a upsamplinng technique. After the transposed convolution, the image is upsized from 28x28x1024 → 56x56x512, and then, this image is concatenated with the corresponding image from the contracting path and together makes an image of size 56x56x1024. The reason for including previous information is to increase the precision. Now it is passed through multiple convolution each of 4 x 4.

This part is repaeted 3 more times as below:

```
u7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (c6)
u7 = concatenate([u7, c3])
c7 = Conv2D(64, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (u7)
c7 = Dropout(0.2) (c7)
```

```
c7 = Conv2D(64, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c7)

u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c7)
u8 = concatenate([u8, c2])
c8 = Conv2D(32, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (u8)
c8 = Dropout(0.1) (c8)
c8 = Conv2D(32, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c8)

u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c8)
u9 = concatenate([u9, c1], axis=3)
c9 = Conv2D(16, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (u9)
c9 = Dropout(0.1) (c9)
c9 = Conv2D(16, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c9)
```

Now the last step is to reshape the image to satisfy our prediction requirements.



```
outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)
```

The last layer is a convolution layer with 1 filter of size 1 x 1 . And the rest left is the same for neural network training.

Machine Learning Unet Neural Networks Deep Learning Convolutional Network

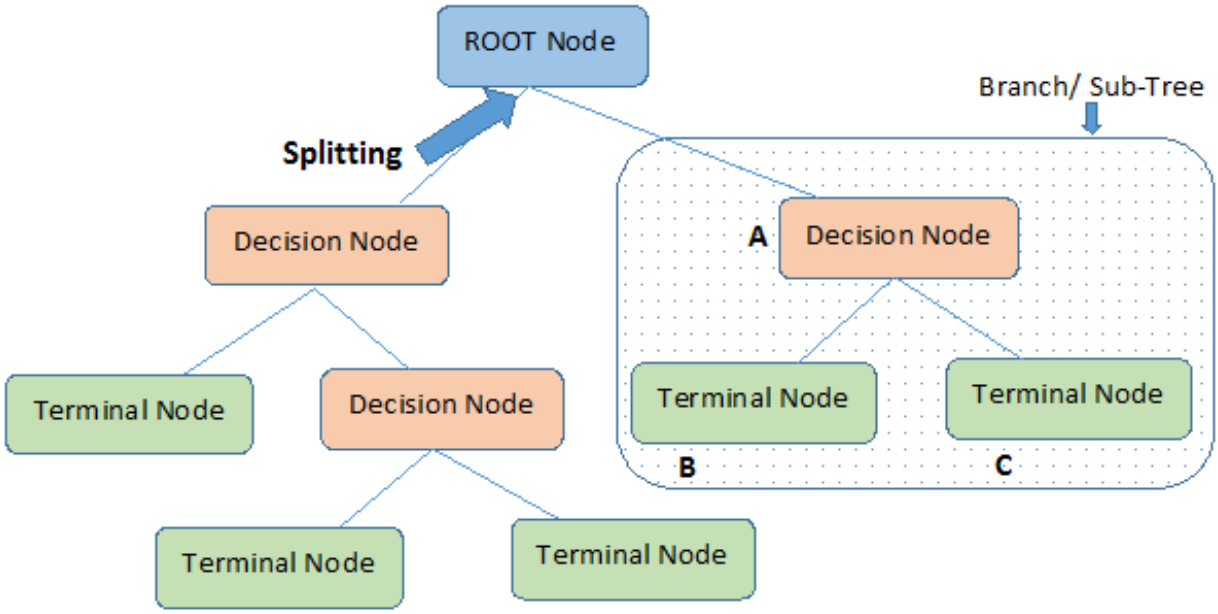
Follow

## Written by Arun Mohan

151 Followers · Writer for DataDrivenInvestor

Machine Learning | AI

## More from Arun Mohan and DataDrivenInvestor



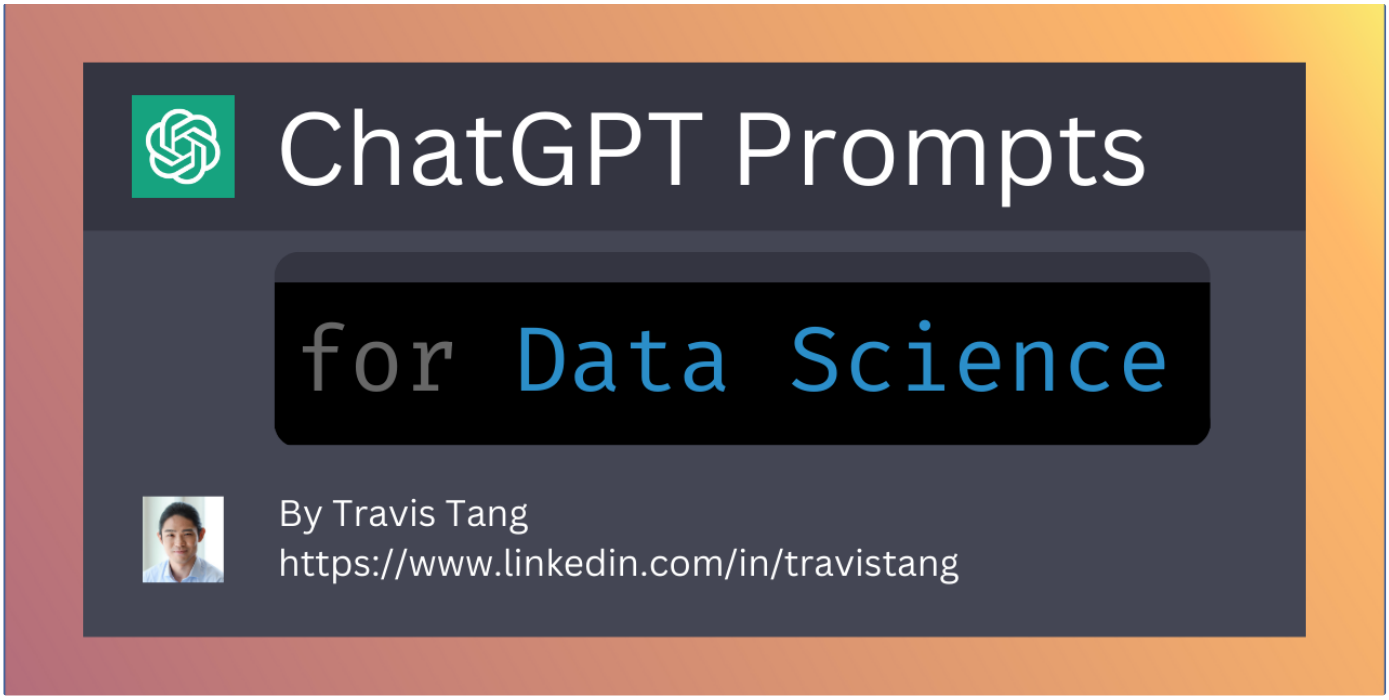

Arun Mohan in DataDrivenInvestor

### Decision Tree Algorithm With Hands On Example

Decision tree is one of the most important machine learning algorithms.It is used for both classification and regression problems.In this...

6 min read · Jan 23, 2019

811 5



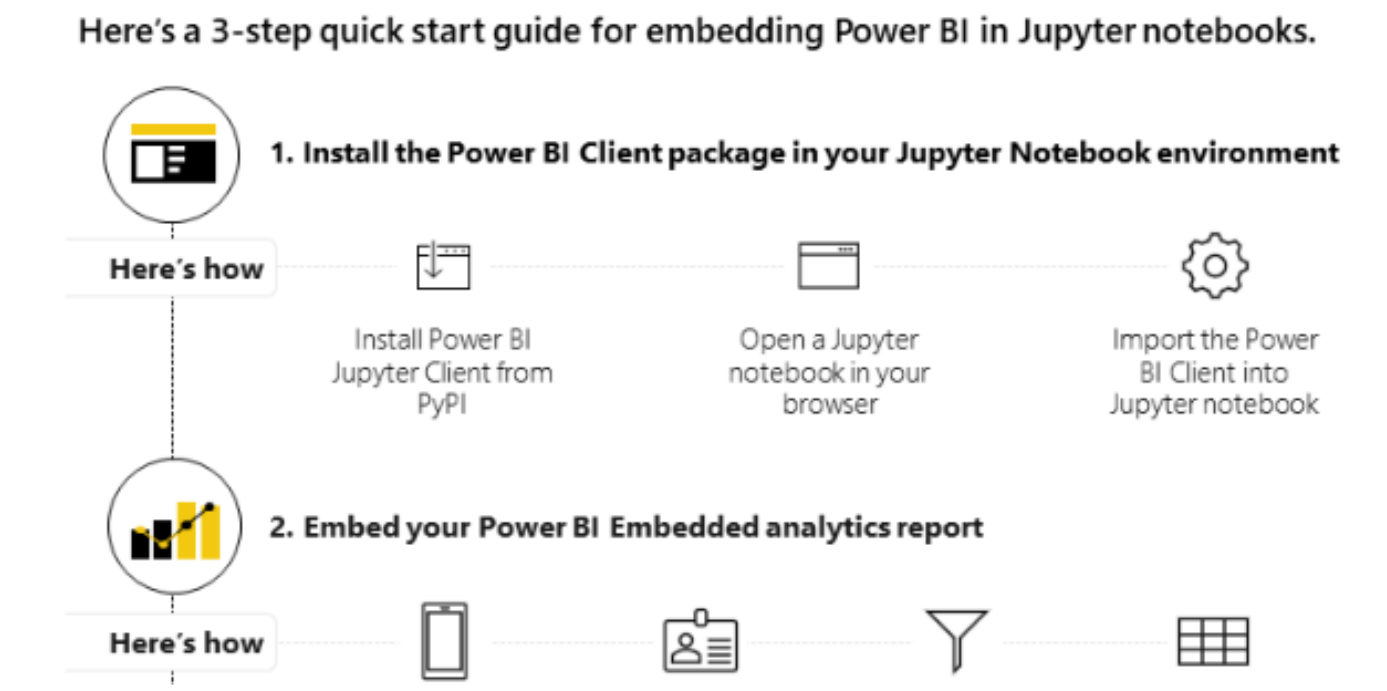

Travis Tang in DataDrivenInvestor

# 60 ChatGPT Prompts for Data Science (Tried, Tested, and Rated)

Automate data science tasks with ChatGPT

27 min read · Apr 11

1.1K 14



Gabe Araujo, M.Sc. in DataDrivenInvestor

## Power BI now integrates with Jupyter Notebooks!

Hello there! Today, I want to share some exciting news with you: Power BI now integrates with Jupyter Notebooks! 😍 📊 📈 🔥 As someone who...

✨ · 8 min read · May 4

705 4

Arun Mohan in Towards Data Science

## IEEE-CIS Fraud Detection - Top 5% Solution

Top 5% solution for Kaggle competition IEEE-CIS Fraud Detection

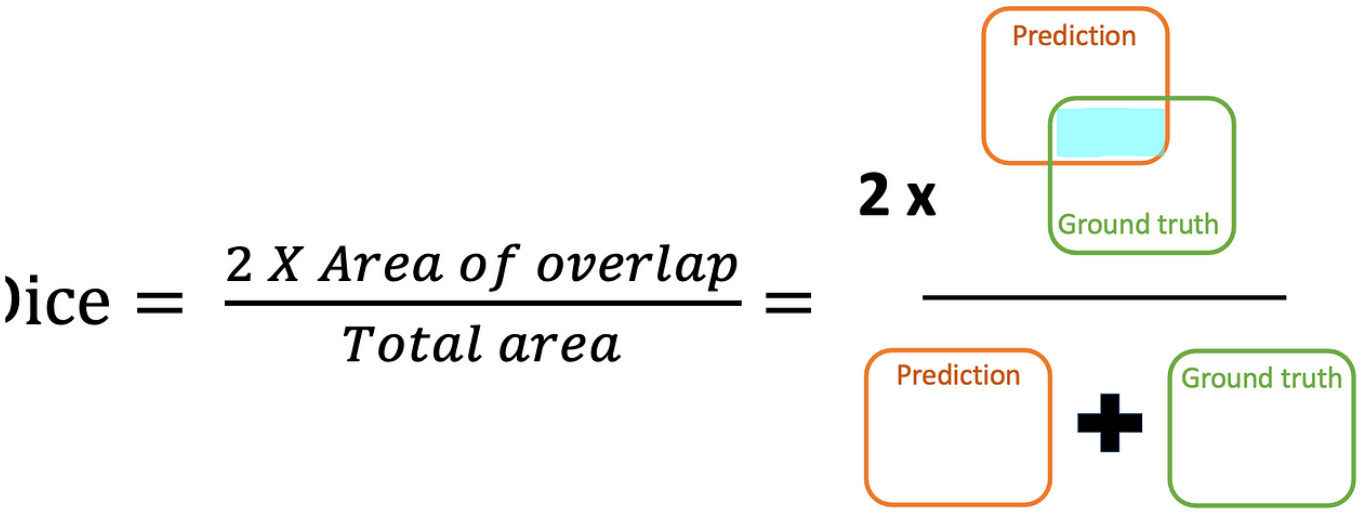
16 min read · Apr 6, 2021

51 1

See all from Arun Mohan

See all from DataDrivenInvestor

## Recommended from Medium




Nghi Huynh in MLearning.ai

### Understanding Evaluation Metrics in Medical Image Segmentation

Implementation of some evaluation metrics in Python

6 min read · Mar 1

115

Enrico Randellini

# Image classification: ResNet vs EfficientNet vs EfficientNet_v2 vs Compact Convolutional...

Fine-tune and compare the latest deep neural network architectures to perform image classification tasks with PyTorch

12 min read · Jan 5

58 3

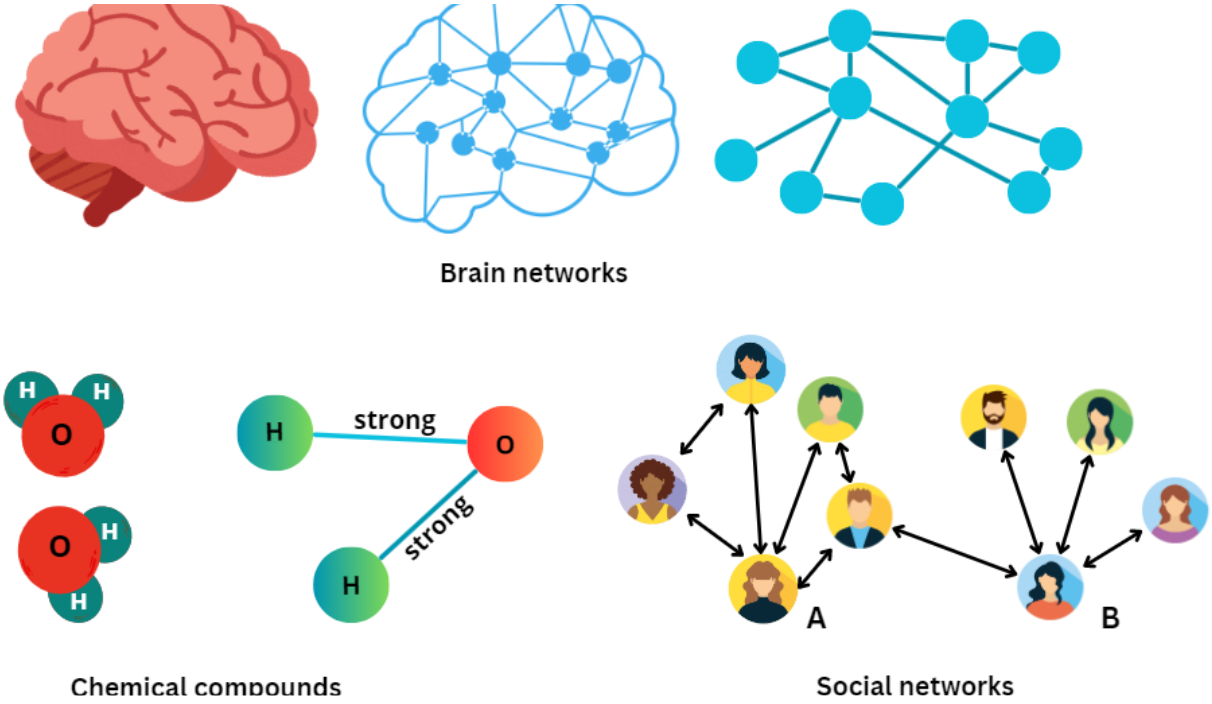
## Lists

**What is ChatGPT?**

9 stories · 52 saves

**Staff Picks**

323 stories · 81 saves



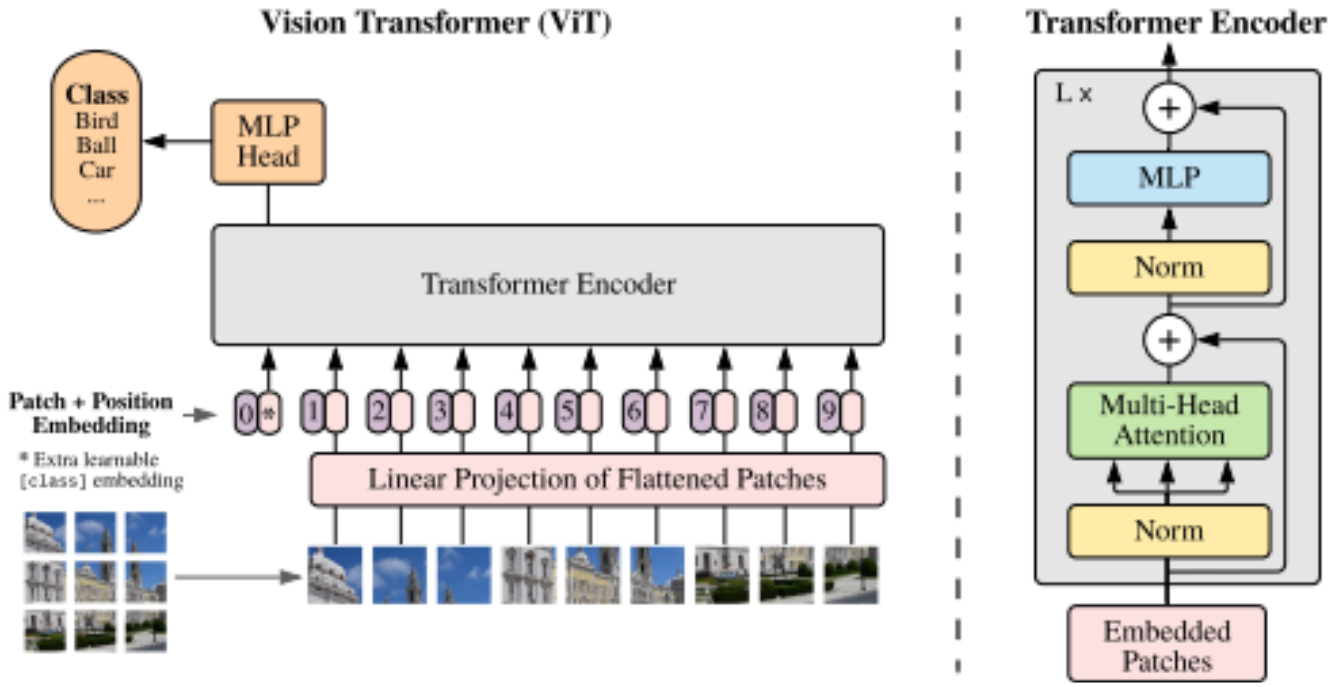

Bscarleth Gtz

## Introduction to Graph Neural Networks: An Illustrated Guide

Hi Everyone! This post starts with the basics of graphs and moves forward until covering the General Framework of Graph neural networks...

14 min read · May 17

30



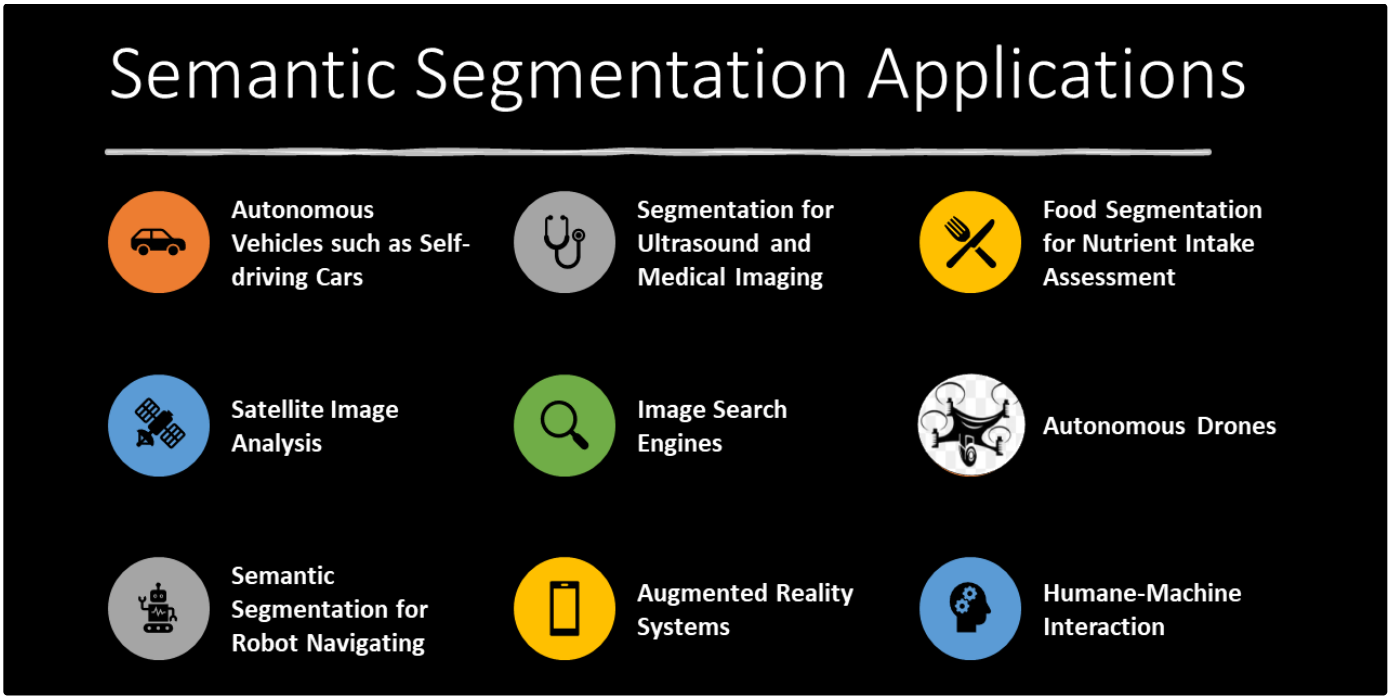

Vikram Pande

## CNNs and Vision Transformers: Analysis and Comparison

Exploring the effectiveness of Vision Transformers and Convolutional Neural Networks (CNNs) in image classification tasks.

6 min read · Apr 29

80 3




Mazhar Hussain

## Semantic Image Segmentation with Python & Pytorch

Semantic segmentation is a computer vision task that involves classifying every pixel in an image into predefined classes or categories...

5 min read · Feb 14

7