**Chapter 6**

**Problem 6: Lesion Segmentation**

It is well known that imaging has a long history in medicine. Technicians utilize images to understand the functions of organs, check health conditions, analyze how diseases develop and recover, and examine the inner parts of the body. Medical images are used in the diagnosis of diseases, where doctors detect infections or affections. Machine learning models assist doctors in diagnosing diseases in the following two ways:

* Classification: As a result of this method, disease conditions can be predicted. The goal is to determine if an organ is infected or not by analyzing the image. By observing lung scans of patients, the model is capable of recognizing covid-19 infections, recognizing whether or not the patients' lung scans are infected.
* Segmentation: The segmentation technique predicts the location of an infection. It is used to detect infected areas. For example, the model can detect tumors by predicting masks on MRIs in order to recognize their existence as well as their location.

In this problem, we will use CT scan images to determine the location of a lesion. To decide whether surgery is necessary, it is necessary to determine the location of the lesion.

**About Dataset**

This problem consists of two groups of images. The first image is a gray-scale CT scan image of the patient's lungs, called the frame. This image has 512 pixels wide and 512 pixels high. The second image has the same dimensions. 255 or 0 are the values of pixels in the mask images, 255 for pixels located within the lesion area, and 0 for pixels outside of the lesion area. The number of images is 2729.

**­­Introduction**

Unlike typical computer programs, Machine Learning techniques will literally learn from data. Machine Learning algorithms can actually find insights and data even if they are specifically instructed on what to look for in that data, and that's what separates a Machine Learning algorithm from a typical computer program. You're just giving the Machine Learning algorithm a set of rules to follow. Instead of actually telling it what to look for, it will find the insights on its own.

**Why do we use Machine Learning to solve mechanical problems?**

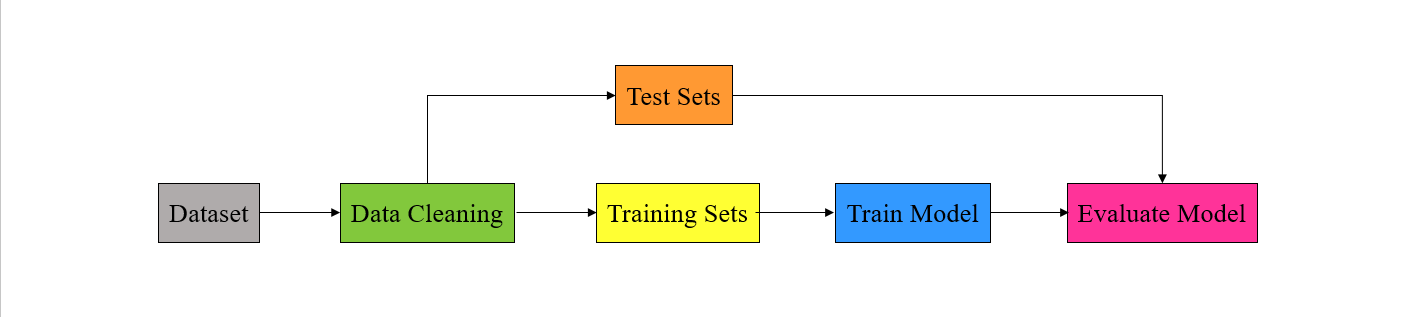
Machine learning is a method for predicting behavior or classifying data sets that, unlike common methods in mechanics, instead of being based on an intuitive model, uses a mathematical model and arbitrary functions to describe and predict the behavior of systems. In other words, machine learning is a search in the space of algorithms and parameters in such a way that it infers a model from the data (data-driven model) and based on that, predicts or categorizes the studied system. For example, using neural networks as one of the methods of traditional machines, I can perform a set of inputs based on an arbitrary number of intermediate hidden layers to the output image results. In the input and output data that are entered quantities, the middle layers do not necessarily have meanings and other adverbial expressions on them. For this reason, I can choose the number of intermediate layers and the number of nodes in each layer at will, and this approach is completely acceptable in the input to the output image. In particular, the relationship between the data is so complex that the models created with a limited number of adjustable settings express this relationship with sufficient accuracy, the efficiency of the methods using machines can be very important.

**Segmentation Problems**

As part of machine learning at a high level, supervised and unsupervised learning methods are used. Supervised learning refers to labeling historical data and using it to inform our models. This label or something we wish to predict is called the target. In supervised learning, there is a specific goal (target) for the past information, whereas in unsupervised learning, there is no specific goal. We use classification and regression in supervised learning. In classification problems, the goal is to determine which category it belongs to. It is usually True or False, but there may be multiple categories as well. In regression problems, a numerical value is our target. A segmentation problem is a classification problem, but the ground truth label and prediction are multidimensional rather than single-dimensional. Our task in image segmentation is to solve a binary classification problem for each pixel in the image.

**What are we going to do in this Chapter?**

We have a dataset from the Kaggle website and then clean that data. After that, we split our data into two groups (train and test). Then we train our model on the training set and after that, we evaluate our model with the test set we have.



***Figure 3-3.***

**TensorFlow**

This open-source library uses tensors to support machine learning and numerical computation. TensorFlow can be developed even by people without any programming experience and can be used in a variety of programming languages, such as Python, JavaScript, and C++.

A major application of TensorFlow is the construction of neural networks such as CNNs and RNNs. TensorFlow, since it is based on graphs, can be executed on multiple processors such as GPUs much more efficiently.



**TensorFlow Layers**

The Keras models, whether sequential or functional, require layers to be defined. Each layer has various parameters that need to be defined. The following layers are used in this notebook:  
  
 **Input:** The input layer, as its name suggests, should be the first layer to receive input. There is only one parameter, input shape, which indicates the shape of input tensors except the batch size.  
   
 **MaxPool2D:** some features in image are not necessary to process. In order to reduce the calculations, after convolution layer, max pool layer pool out important features of the image. For this layer, only the filter size needs to be defined.  
  
 **Conv2D:** By using this layer, hidden features in images can be extracted by performing convolution operations on 2D tensors. Each convolution layer has its own set of parameters, which include the number of filters, filter size, padding, etc.  
   
 **UpSampling2D:** Unlike pooling, this layer generates numbers in order to expand the dimension of the input tensor, rather than pooling out important features. Similar to pooling, only filter size must be specified.   
  
 **Batch Normalization:** It scales input values between 0 and 1 in order to reduce computation load.



**Matplotlib**

The Matplotlib package provides static and dynamic visualization in Python. Pyplot is a Matplotlib module that creates a separate figure and makes changes to this figure to generate a plot.



**NumPy**

The NumPy library contains functions that deal with arrays, which have the advantages of being processed faster than lists, consuming less memory, and reserving a static portion of memory.



**OS**

The Python OS module contains libraries and functions that can be accessed from the underlying operating system to manipulate directories, create and change directories, access files in directories, etc. It is essential when dealing with files, such as reading or writing them.



**Glob**

In Python, the glob module returns a list of all files in a directory with a specific pattern in their name. It is frequently used to access local files.

****

**Keras Backend**

In TensorFlow, all operations are performed on graphs, and the model must understand any relationships between variables and operations. Keras's backend is a useful tool for defining custom loss functions, metrics, and custom arithmetic operations.

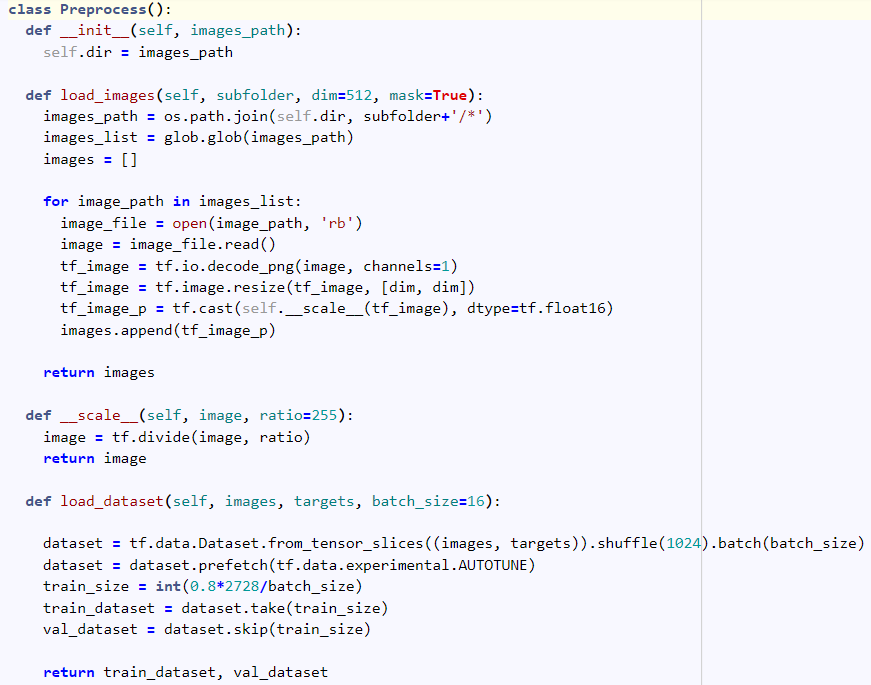


**Step 1. Preprocessing**

Preprocess is a class that contains four methods for preparing data for training. The first method takes care of initializing the class by storing the path of the image folder as an attribute. The second method loads images from a specific directory. Join the group name to the image directory, then save a list of all images into the images\_list variable. To load the images individually, a loop through each image is required.

Each image path in the list of image paths is opened and read by Python functions. This is followed by the decode\_png function of TensorFlow to save the image as a tensor. Next, a resize method is responsible for changing the dimensions of Images to a customized one. The image pixels are scaled between 0 and 1. Finally, the image is added to a list of images.

The third method scales the tensor values, while the last method creates a dataset using input lists containing masks and frames. The first step is to create a dataset variable using TensorFlow's method from\_tensor\_slices, shuffle it in a buffer of 1024, and then divide it into batches of a specific size. The dataset is then fetched by an automatic buffer at the next stage. Lastly, the predefined dataset is divided into two datasets, training and validation, with an 80-20 ratio.



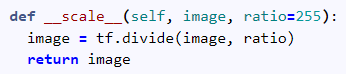
**Normalization of Data**

There are varying ranges of numerical values in machine learning, in order to train faster and more consistently, these values should have a similar range, such as between 0 and 1. This type of processing is referred to as normalizing because all values scale between 0 and 1.

It is possible to use pixel values as features in this problem. Pixel values are eight bytes long and are between 0 and 255 in size. In order to achieve a range between 0 and 1, pixel values must be divided by 255, but before returning values, the result of the division must be cast as a float to prevent further TensorFlow errors (casting means changing value types).

The table below illustrates some random data between 0 and 10. To normalize the data, subtract each number from the minimum (0) and divide it by the range (10 - 0).

|  |  |  |
| --- | --- | --- |
| *X* |  |  |
| 4 | (4 - 0)/(10 - 0) | 0.4 |
| 10 | (10 - 0)/(10 - 0) | 1 |
| 0 | (0 - 0)/(10 - 0) | 0 |
| 5 | (5 - 0)/(10 - 0) | 0.5 |
| 7 | (7 - 0)/(10 - 0) | 0.7 |

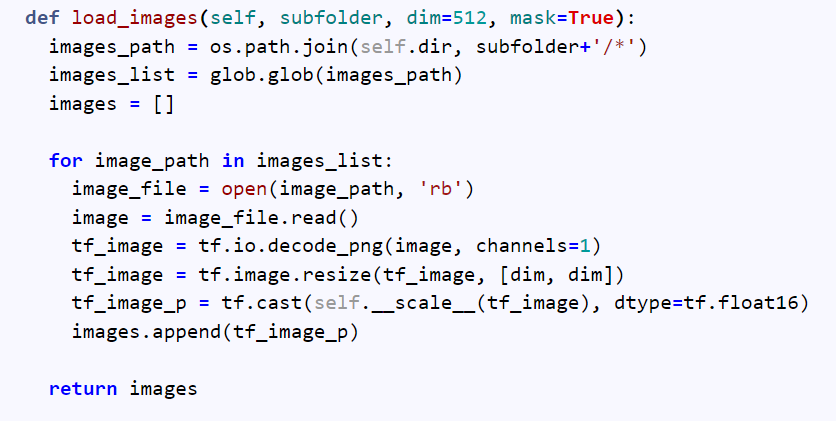
****

**Why Do We Need to Normalize Data in Python?**

One of the most important topics in the field of Machine Learning and Data Mining, especially in the Data Preparation section, is the topic of Re-scaling of data, which is usually done by There are two methods of Standardization and Normalization. The meaning of normalization is to transform the data into the domain [0 and 1]. Each of the data recorded in the dataset will change to a range between zero and one. This makes the data fall under a shorter domain and the model is trained better.

**Step 2. Load Data**

In order to reduce the amount of RAM usage, dimension of images are decreased to 128 and batch size is set as 4.

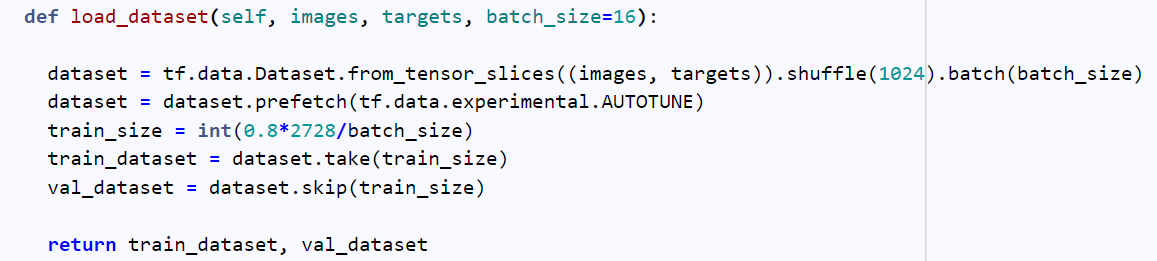


**Step 6. Split Training and Testing Datasets**

Data is the key to training all machine learning models, and parameters are changed based on the available data. Therefore, more data results in better results. Using gradient descent, the model calculates training data loss and attempts to decrease the loss value. However, does a decreasing training data loss mean better results on real-world data? Not necessarily. Real-world data often include data that the model has never encountered previously. Thus, for evaluating results, loss values on training data are not sufficient, and some additional data, such as validation, must be collected.

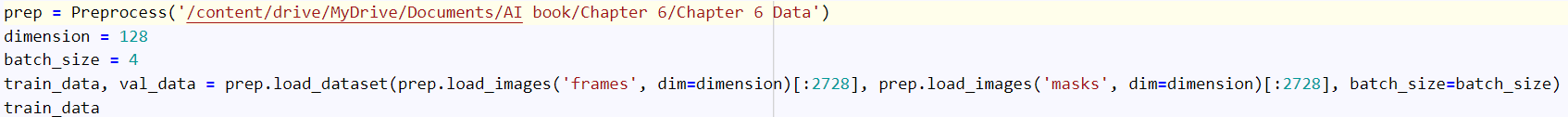
It is very critical to determine the number and size of validation data, and this selection is highly dependent upon the number and size of data. Despite the fact that it is not a principle, it is acceptable to split 20 percent of the data between validation and training. The distribution of validation data must be the same as the distribution of actual data in order to achieve better results.

To divide a dataset into training and validation, I use the take and skip methods. In this example, 80 percent of the data is set aside for training and 20 percent for validation. The Take method accepts a number that determines the number of batches to split for data. Skip method takes the same number in order to ignore the first batches as much as you specify.

****

**Batch**

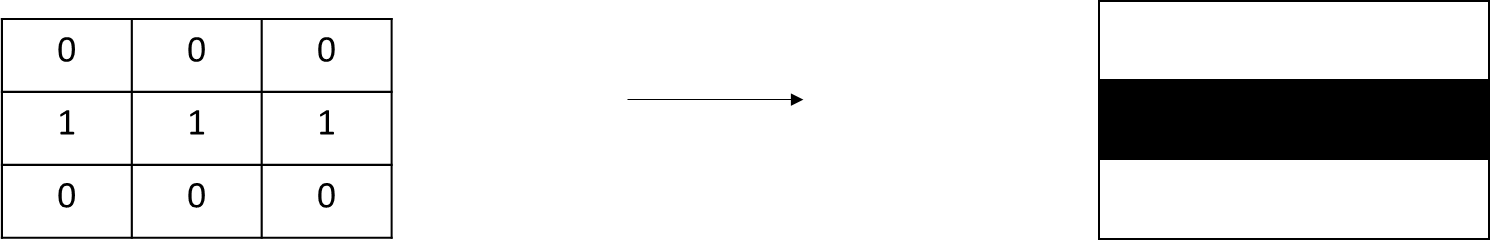
It is much more computationally efficient to compute loss on a small portion of data and calculate gradient descent based on that rather than computing loss on the whole data. A batch is a set of data that is used to calculate loss once a time and a batch size is the number of data in each batch. Batch size is the first element of the shape in Tensorflow and it is saved as a None object since it is independent of the model architecture.

****

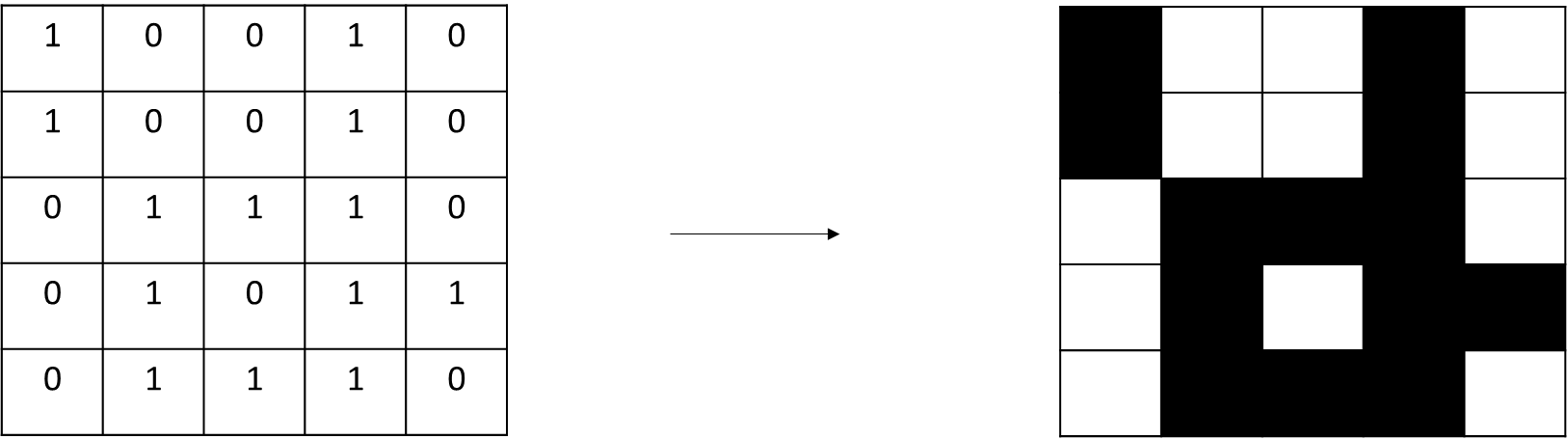
**Step 7. Training**

**Convolution**

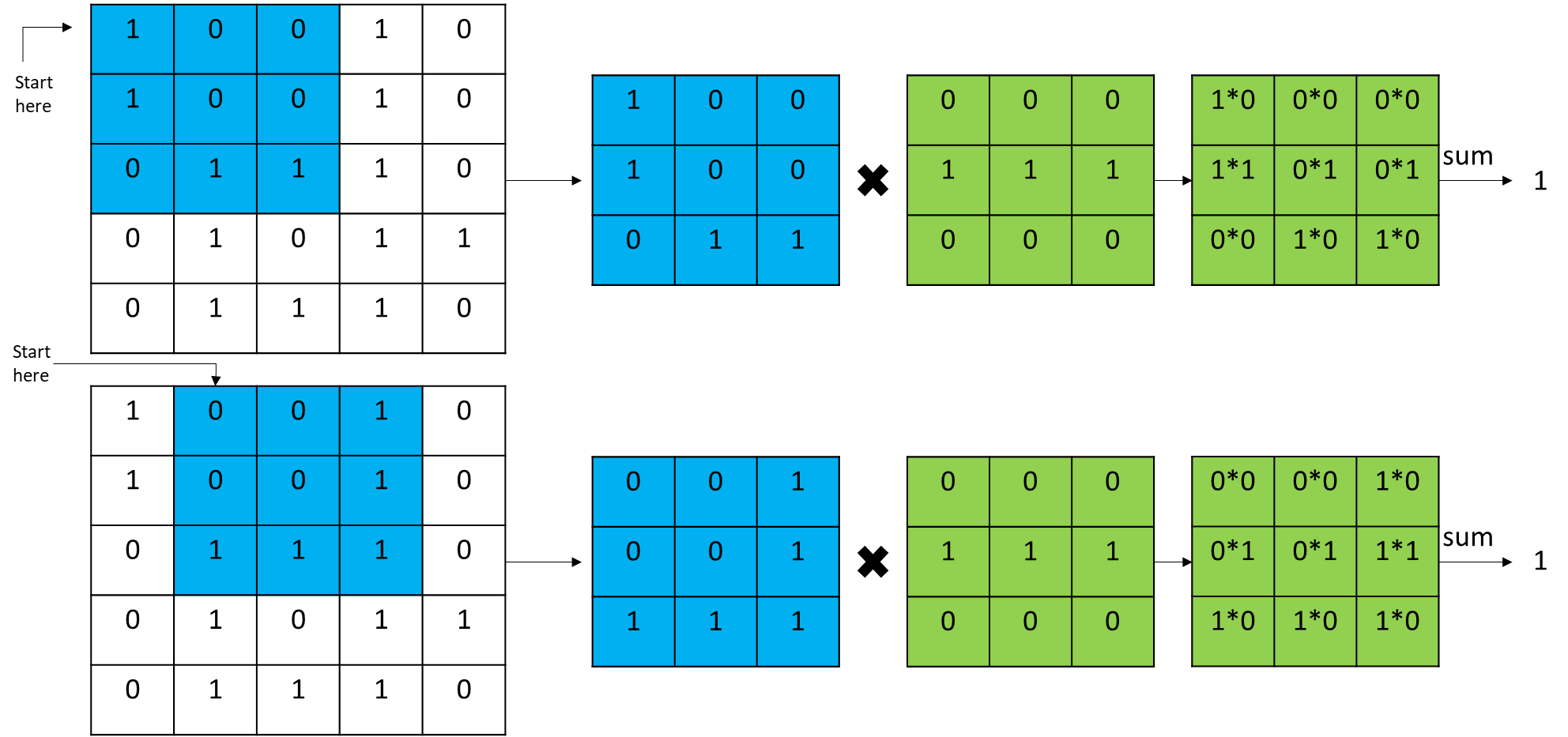
Convolution is an operation that is specifically designed to extract useful information from images. Essentially, this operation is based on human vision and how the brain detects specific patterns in images. In order to extract hidden features from an image, a special filter must be applied to the image. For example, this filter can be used to detect horizontal lines.



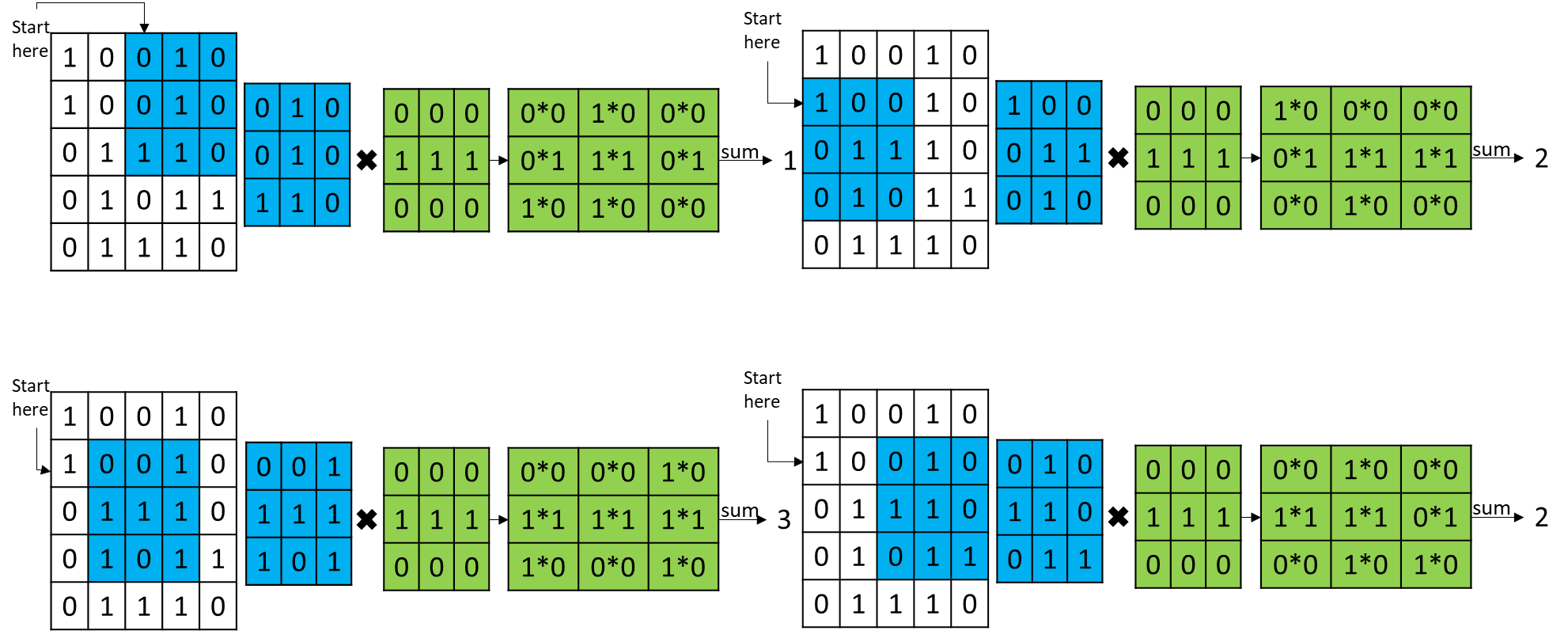
As can be seen, this filter has a horizontal line within itself. In order to clarify the convolution operation, let's mention an example and compute convolution on it.



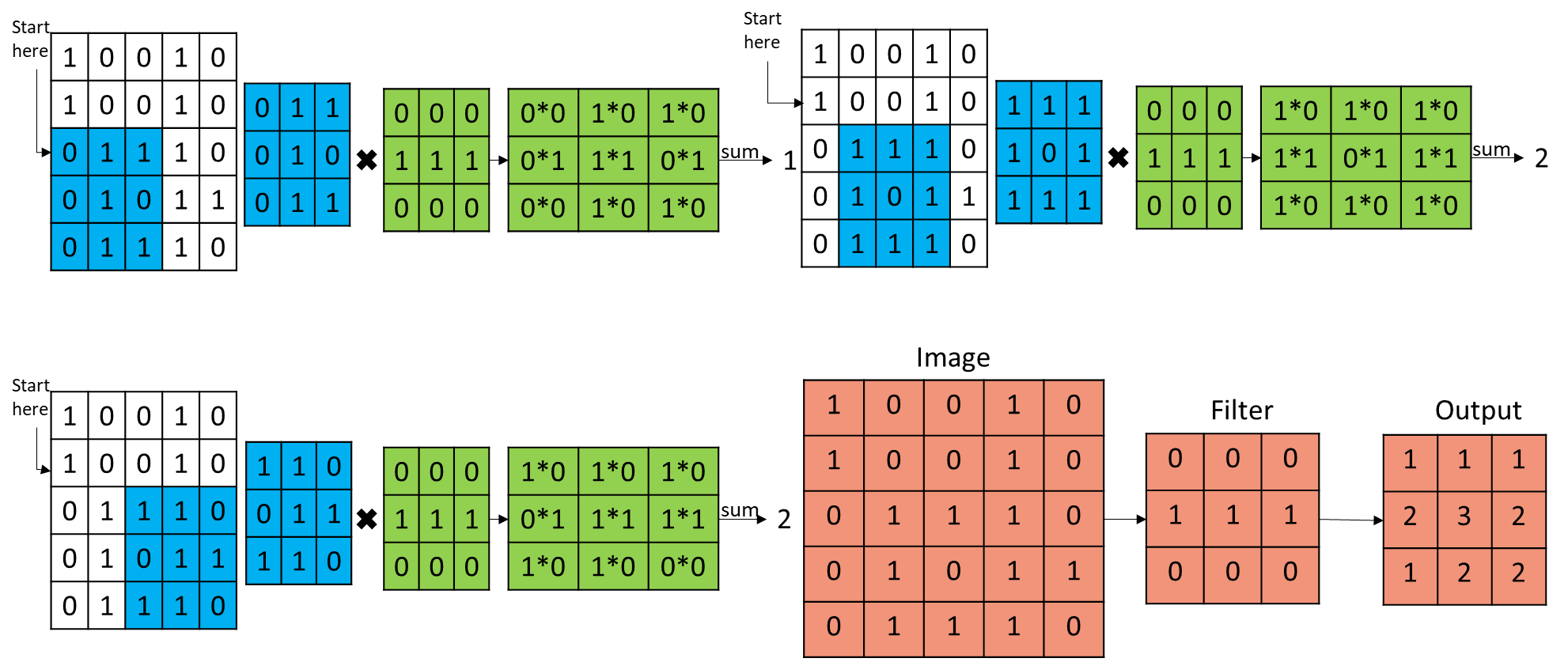
To perform a convolution operation, the elementwise product of the most upper left part of the image and filter is computed, then the next part of the image is computed, and so on.



The process continues until the end is reached.

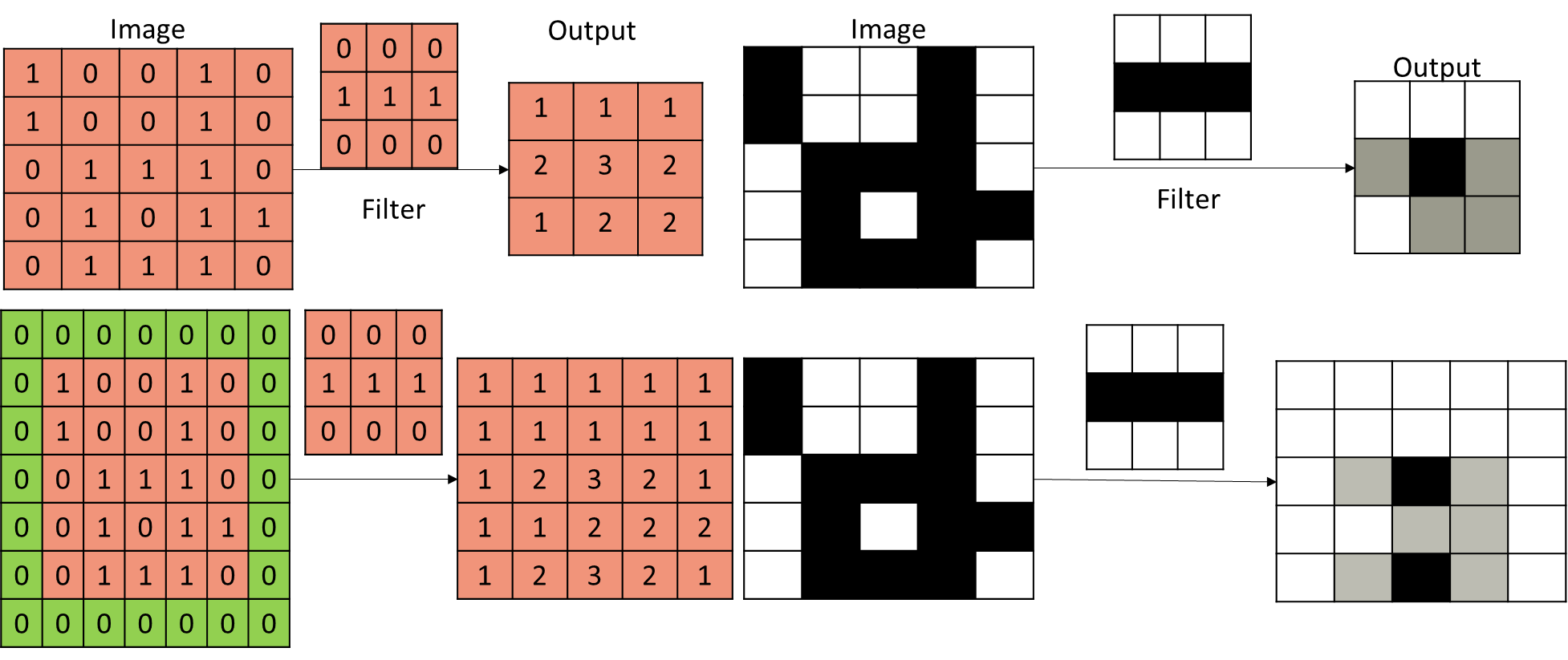
****

In the end, all nine computed elements are positioned in their respective positions.

****

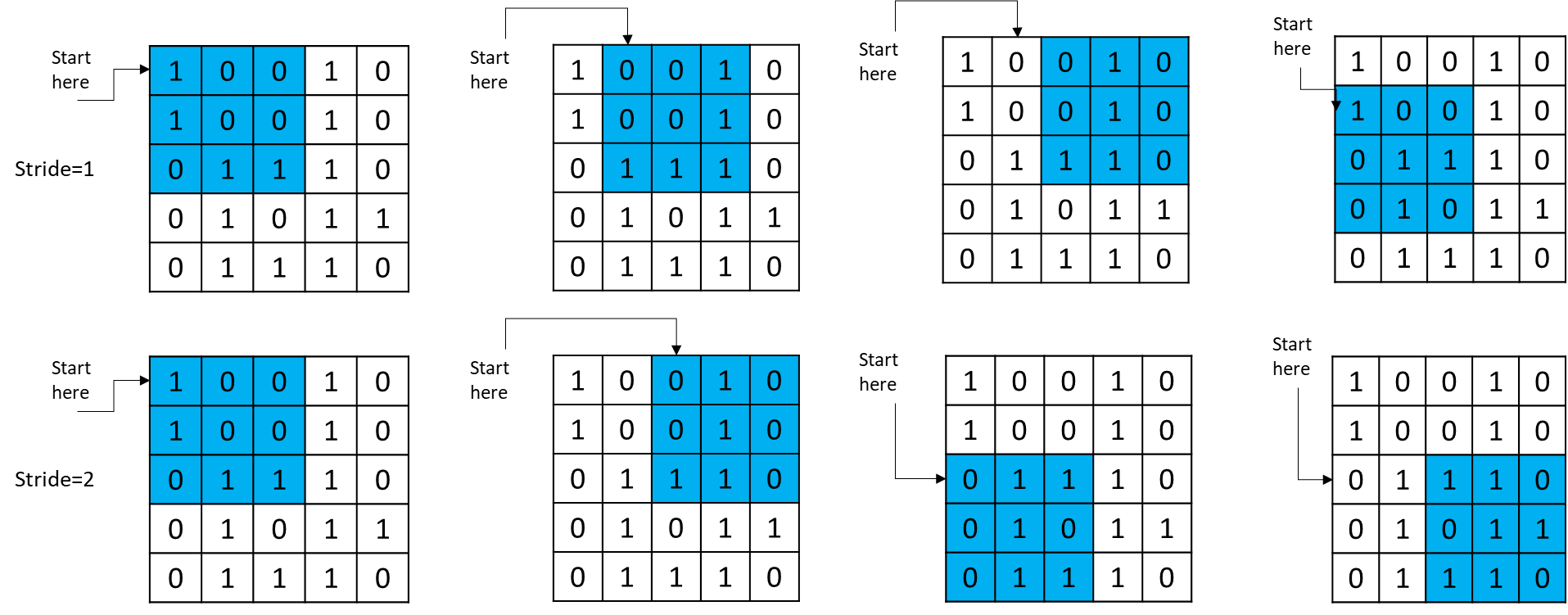
**Padding**

Clearly, the filter detects only the middle horizontal line (the middle element of the output is black). The horizontal line on the edge is not detected by the filter. The dimensions of the image are increased by two and empty elements are replaced by zero in order to detect features on the edges.



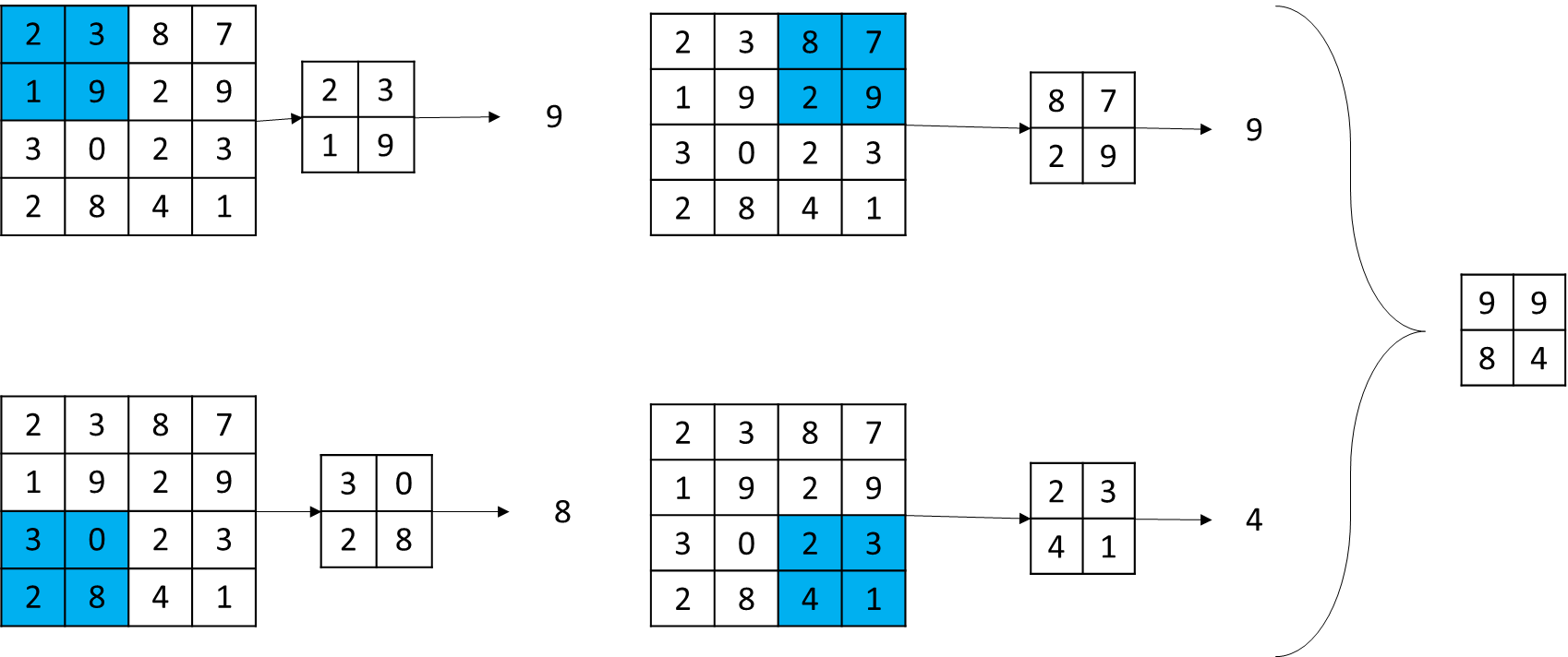
**Stride**

A stride is the step taken by the filter in each convolution operation. In the first example, stride is 1. In the second example, stride is 2.



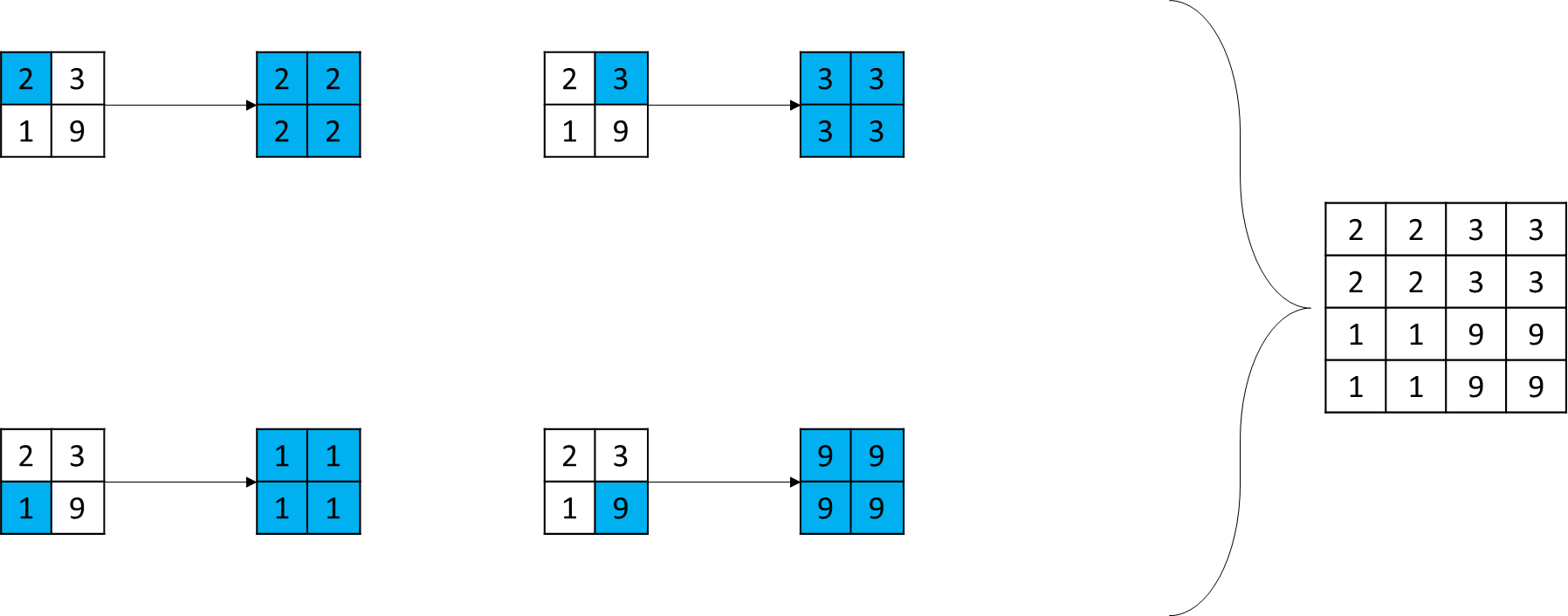
**Max Pooling**

The max pooling function extracts the largest value from the kernel, similar to convolution filters, by importing kernels in each step. The example below shows a 2 by 2 max pooling operation.



**Up Sampling**

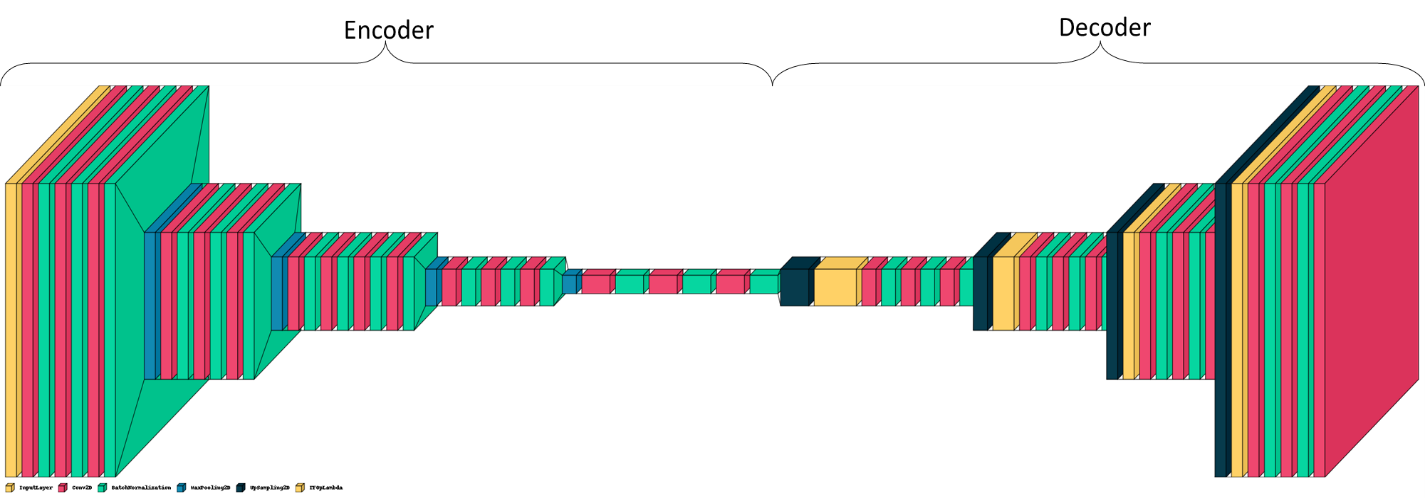
To regenerate the features of the original image, the up sampling layer increases the image's dimensions. Each step is performed by importing kernels. The example below shows a 2-by-2 up sampling process.



**Create Model**

In this function, a neural network architecture is created. The first layer is the input layer. The input shape (4, 128, 128, 1) refers to a grayscale image of 128 by 128 pixels.

In architecture, there are two main components. The first is the encoder, which extracts useful information from the image by decreasing its dimension. The second part is the decoder, which attempts to generate features by mapping latent space features to an image of the same size as the input.



**def** neural\_net**(**input\_shape**=**(128, 128, 1), batch\_size**=**4**)**:

  inputs **=** Input(shape**=**input\_shape, batch\_size**=**batch\_size)

  x **=** Conv2D(32, (3, 3), 1, activation**=**'relu', padding**=**'same')(inputs)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(32, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(32, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x1 **=** BatchNormalization()(x)

  x **=** MaxPool2D((2, 2))(x1)

  x **=** Conv2D(64, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(64, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(64, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x2 **=** BatchNormalization()(x)

  x **=** MaxPool2D((2, 2))(x2)

  x **=** Conv2D(128, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(128, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(128, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(128, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x3 **=** BatchNormalization()(x)

  x **=** MaxPool2D((2, 2))(x3)

  x **=** Conv2D(256, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(256, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(256, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x4 **=** BatchNormalization()(x)

  x **=** MaxPool2D((2, 2))(x4)

  x **=** Conv2D(512, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(512, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(512, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** UpSampling2D((2, 2))(x)

  x **=** tf.concat([x4, x], axis**=**-1)

  x **=** Conv2D(256, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(256, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(256, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** UpSampling2D((2, 2))(x)

  x **=** tf.concat([x3, x], axis**=**-1)

  x **=** Conv2D(128, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(128, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(128, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** UpSampling2D((2, 2))(x)

  x **=** tf.concat([x2, x], axis**=**-1)

  x **=** Conv2D(64, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(64, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(64, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** UpSampling2D((2, 2))(x)

  x **=** tf.concat([x1, x], axis**=**-1)

  x **=** Conv2D(32, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

  x **=** BatchNormalization()(x)

  x **=** Conv2D(32, (3, 3), 1, activation**=**'relu', padding**=**'same')(x)

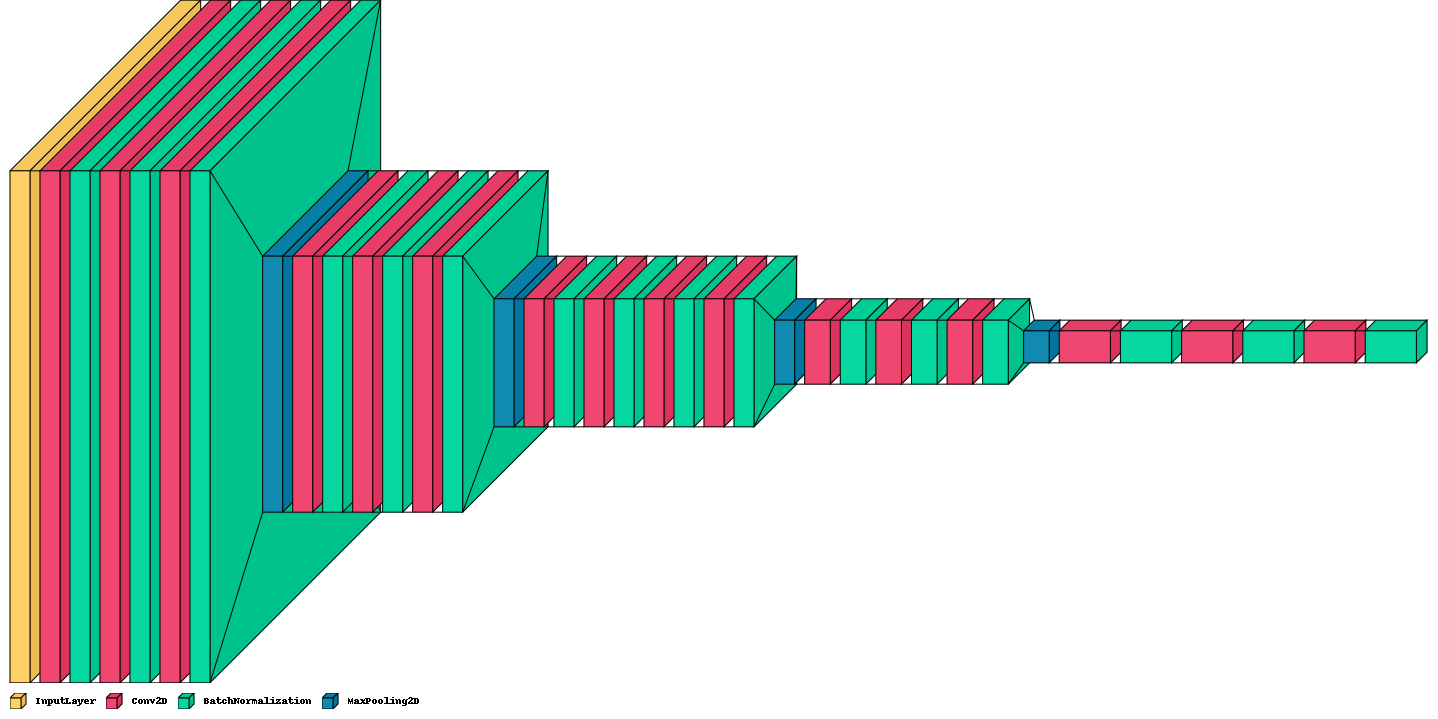
  x **=** BatchNormalization()(x)

  x **=** Conv2D(1, (3, 3), 1, activation**=**'sigmoid', padding**=**'same')(x)

**return** tf.keras.Model(inputs**=**inputs, outputs**=**x)

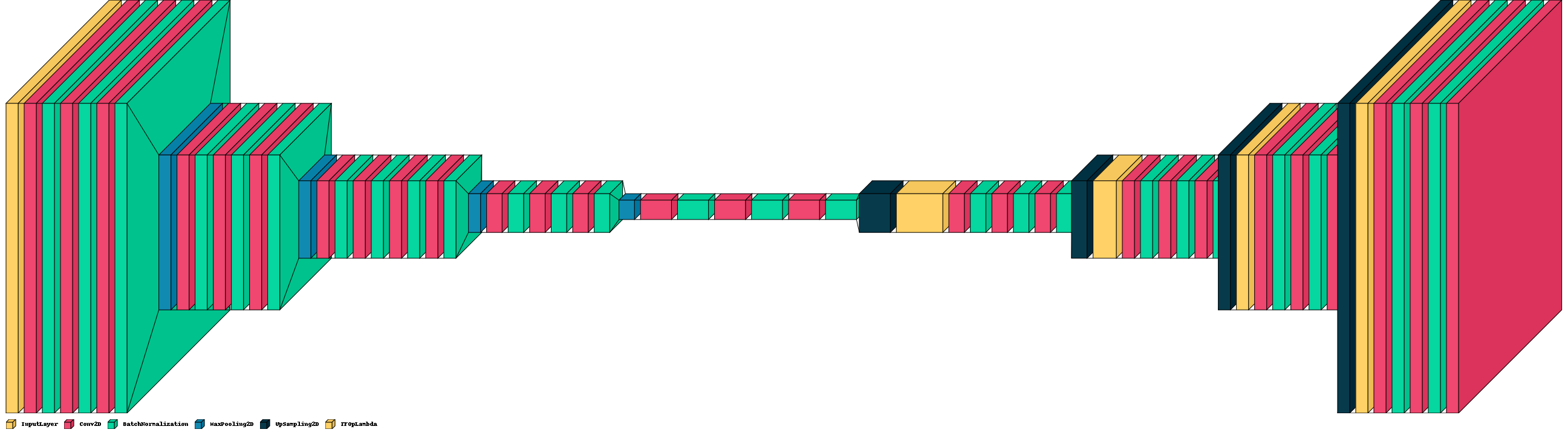
**Encoder**

The encoder part is composed of 5 different convolution blocks. Each block consists of 3 convolution layers in order to extract features, as well as a max pooling layer to highlight key features. In the encoder, the filter size of the convolution layers decreases as it progresses through the process to recognize more minimal features. However, the number of filters increases in order to enhance the depth and details of the image. In order to ensure smooth convergence, batch normalization is implemented after each convolution layer.

****

**Decoder**

The decoder part is composed of 4 different convolution blocks. At the beginning of each block, an upsampling layer increases the image dimensions by 2. After upsampling, concatenation is performed in order to regenerate the original features of the image. This means that two parts of the model, which have the same dimensions except for the number of channels, are merged together. As a result, the model will remember the original image that was encoded. At the end of block 3 convolution layers attempt to detect and regenerate image features. Through the model, the number of filters decreases in order to regenerate the basic features, but the image dimensions increase in order to reach the initial dimensions.



**Sigmoid**

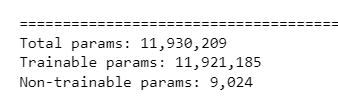
The Sigmoid function converts a number into a probability distribution. Neural networks deal with numbers on various scales. In image segmentation, the model outputs a matrix of numbers with dimensions equal to the original image. All pixel values must be between 0 and 1 in order for this matrix to be represented as an image. This can be done using the Sigmoid function as follows:

**Summary**

In the model summary, we can see the output and number of parameters for each layer, which provides a better understanding of the model. Approximately all parameters are trainable. More than 13 million parameters indicate the enormous amount of learning that has taken place.

During training, all parameters are trainable. As the model continues to learn, it updates its values using gradient descent; however, non-trainable parameters remain fixed during training. The process of setting more trainable parameters requires additional time and computation; however, it completes the learning process.





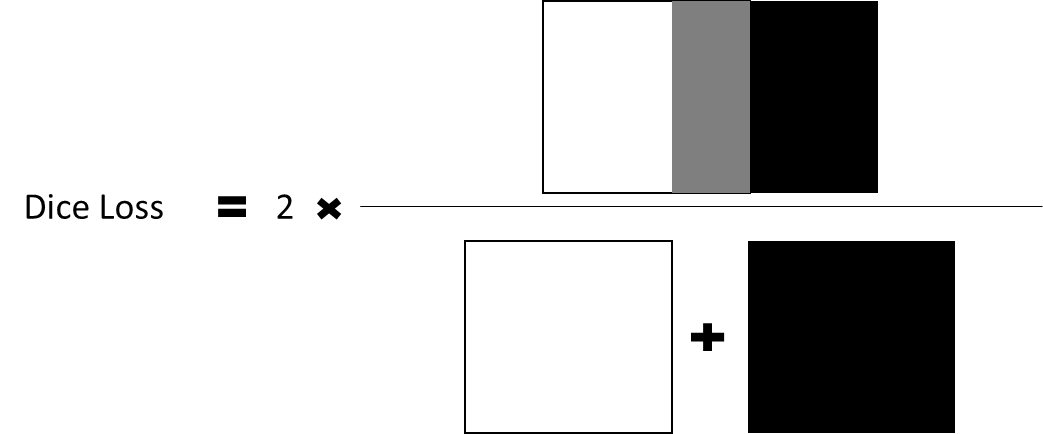
**Compile Model**

Compile model means to define 3 sets of parameters, loss function, optimizer and metrics. Loss function is a function that compute loss. In this example has been set to custom the Dice loss that defined previously.

****

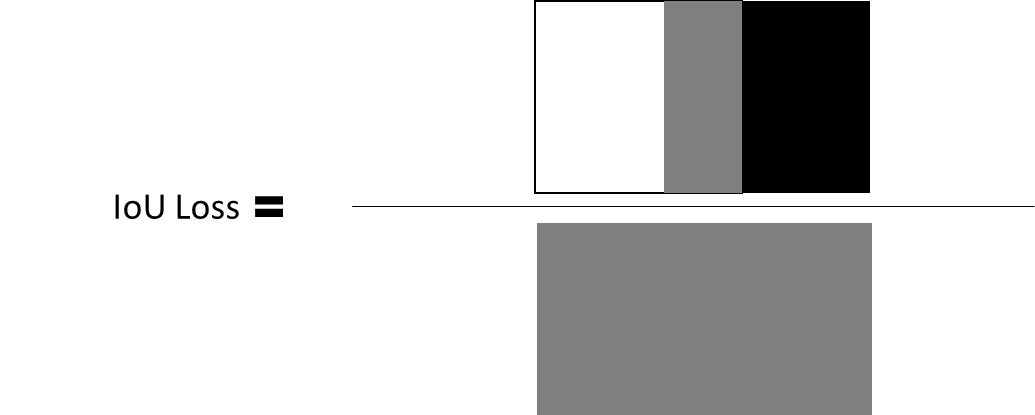
**Dice Loss**

For image segmentation, the output of the model and ground truth label is a matrix representing an image. As a loss calculation method, dice loss is one of the best-defined methods. Using the intersection of the two images, it calculates the loss by dividing it by the sum of the two images and multiplying it by two.



**IoU Loss**

The term IoU loss refers to intersection over union loss and it is very similar to dice loss. IoU loss calculates the intersection of two images and then divides that by the union between the two images to calculate the loss.



**Adam**

Optimizer is a function that can be used to compute gradient descent. It is set to Adam in this example. The SGD algorithm is very noisy, and it does not descend well on curves. To decrease noise in steps moving averages, a new parameter called Momentum is defined. The SGD algorithm with Momentum performs better on curves and requires fewer steps to converge. This method employs two momentum variables, a first-order momentum, and a second-order momentum, as well as an epsilon value that prevents division by zero. Adam is extremely efficient and useful for the convergence of local minima.

**Accuracy**

Metrics has been set to accuracy means that how much model predict accurately. The accuracy formula showed in below and as it suggests to evaluate our model by computing number of times that model predict right over by number of times that model predicts.

**Fit Model**

After all previous steps were completed to prepare the model for training, all that remains is to fit the model over the data. In order for the training model to be fit over the data, training and validation data need to be determined, as well as a number of epochs, where an epoch is the number of times the model was trained.

**Epoch**

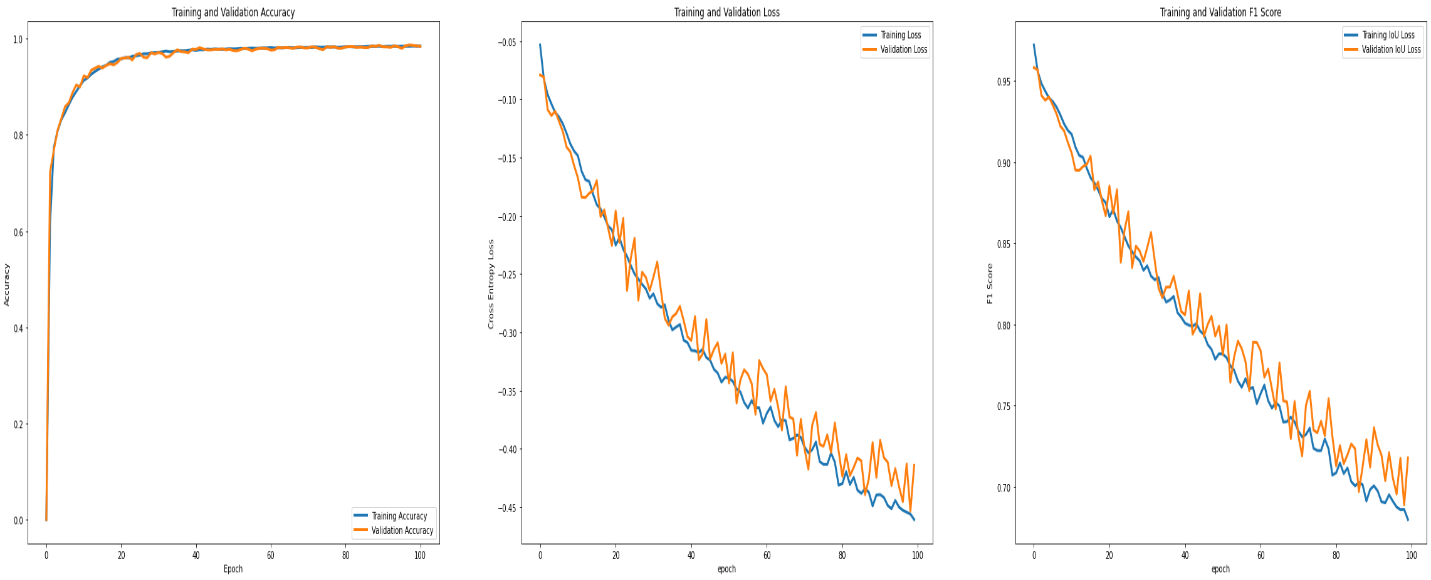
In training, the training process passes through all training data, so the number of training epochs indicates how many times the training process has been passed through all training data. When dealing with batched datasets, the number of times that training algorithm has performed is not important. The number of epochs is an important parameter to control the model since it indicates how many times the loss was computed on the entire data set.

****

**Analysis**

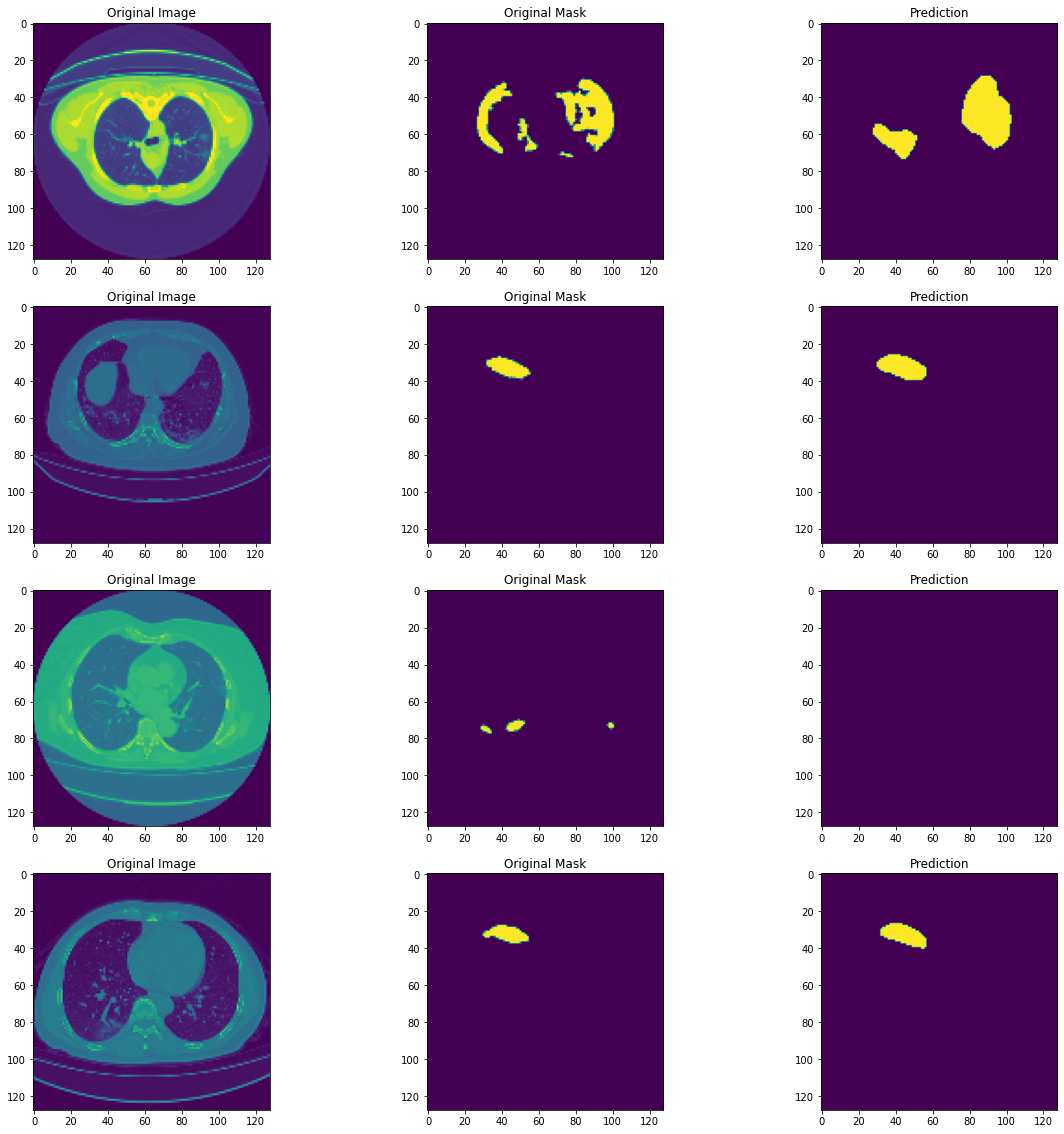
Using training as the number of epochs, it is possible to inspect the parameters of loss and accuracy. It is evident from the loss and accuracy plot that approximately 98% accuracy is acceptable for both validation and training.



****

**Evaluate**

In order to observe the performance of the model, output of the model on the 1 batch of validation data is represented.

****

**References**