(https://analyticsindiamag.com)

PUBLISHED ON SEPTEMBER 4, 2021
IN DEVELOPERS CORNER
(HTTPS://ANALYTICSINDIAMAG.COM/CATEGORY/DEVELOPERS_CORNER/)

Guide to Different Padding Methods for CNN Models

the convolutional layers reduce the size of the output. So in cases where we want to increase the size of the output and save the information presented in the corners

By Yugesh Verma(https://analyticsindiamag.com/author/yugesh-vermaanalyticsindiamag-com/)

Convolutional neural networks(CNNs) are used every day for tackling various problems occurring in image processing and predictive modelling or classification tasks. The most popular application for CNNs is to analyze image data. As we know mathematically, every image in any dataset is a matrix of its pixel values. When working with the simple CNN (https://analyticsindiamag.com/guide-to-text-classification-using-textcnn/) for an image, we get the output reduced in size which we can definitely consider as the loss of data. And sometimes it becomes very difficult to generate a proper result according to our requirements. In that case, where we don't want the shape or our outputs to reduce in size, the addition of more layers in the data can help and that addition can be done by padding.

In this article, we will discuss padding with its importance and how to use it with CNN models. We will also discuss different methods of padding with how they can be implemented. Below are the important points that we are going to cover in this article.

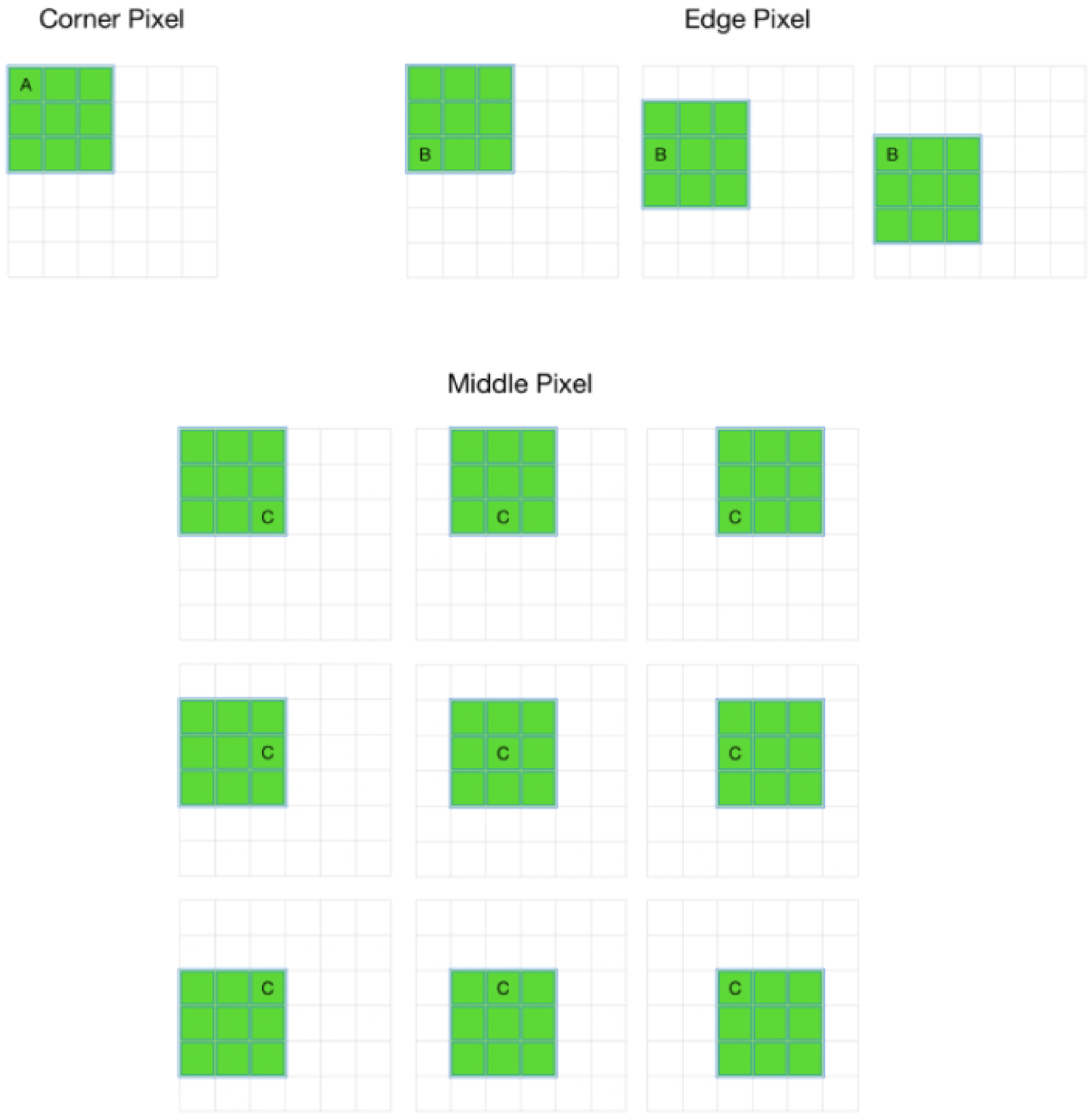
Table of Contents

3. **Types of Padding**
    1. **Same padding**
    2. **Causal padding**
    3. **Valid padding**
4. **How to use Padding in CNN Model**

Let's begin with understanding the problem faced with simple convolutional layers.

Problem with Simple Convolution Layers

A simple CNN gives results For a grayscale image of size (n x n) with (f x f) filter/kernel size is (n – f + 1) x (n – f + 1). For example in any convolution operation with a (8 x 8) image and (3 x 3) filter the output image size will be (6 x 6). So this happens every time when processing images, the output of the layers is shrunk in comparison to the input. Also, the filters we are using do not focus on the corners every time when it moves on the pixels. For example,



The above image is an example of the movement of a filter of size (3 x 3) on an image of size (6 x 6). We can clearly see that a corner pixel A is coming under the filter in only one movement where b is coming in three movements and c is coming under 9 movements. It basically shows that the model will work with pixel C very fine and it will misinterpret pixel A.

This will cause the loss of information available in the corners and also the output from the layers is reduced and reduced information will create confusion for the next layers. This problem of the model can be reduced by the padding layers.

What is Padding?

As we just discussed, the convolutional layers reduce the size of the output. So in cases where we want to increase the size of the output and save the information presented in the corners we can use padding layers where padding helps by adding extra rows and columns on the outer dimension of the images. So the size of input data will remain similar to the output data.

Padding basically extends the area of an image in which a convolutional neural network processes. The kernel/filter which moves across the image scans each pixel and converts the image into a smaller image. In order to work the kernel with processing in the image (https://analyticsindiamag.com/image-processing-with-opencv-in-python/), padding is added to the outer frame of the image to allow for more space for the filter to cover in the image. Adding padding to an image processed by a CNN allows for a more accurate analysis of images.

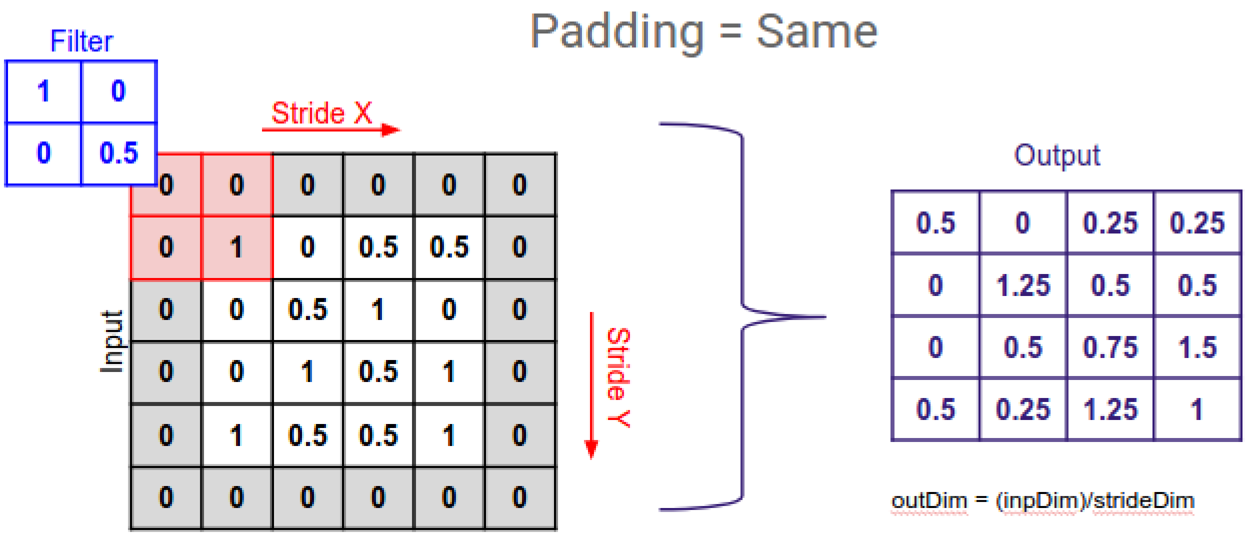


Image source (https://images.deepai.org/django-summernote/2019-05-27/c3f24854-5584-4feb-81d7-3bcc5800a689.png)

In the above image, we added the padding layer(grey color rows and columns) on an image and this is how it saves the size of the image matrix from reducing the size.

Types of Padding

There are three types of padding:

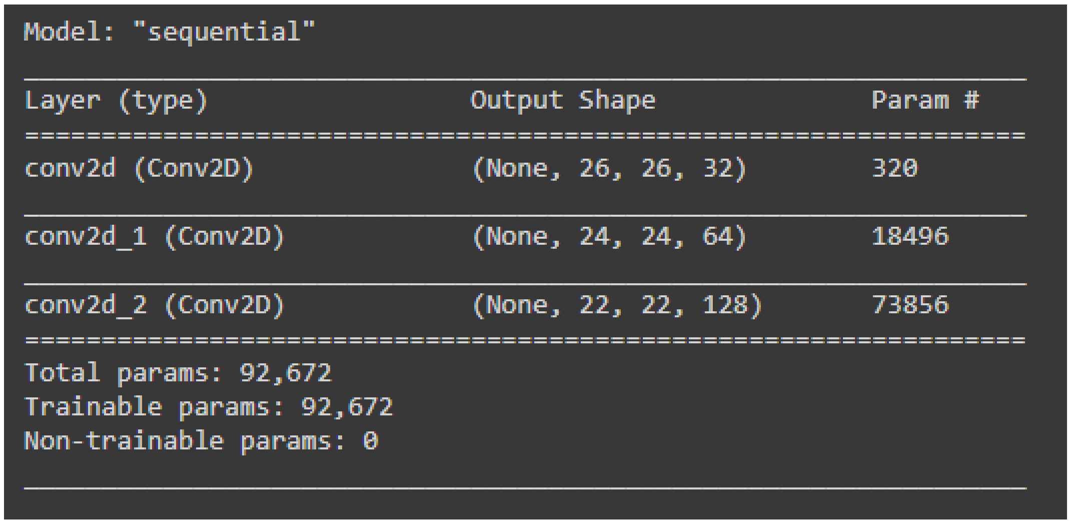
- Same padding
- Causal padding
- Valid padding

Before introducing different types of padding we should start with discussing how we make a model so that we can get a proper difference between a no padding model and a padding model.

Here I am using Keras for building a model which can be fitted into data when required. A simple CNN model starts from a sequential instance. After this instance, we can add some convolutional layers to the model.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(28, 28, 1)))
model.add(Conv2D(64, kernel_size=(3, 3),
activation='relu'))
model.add(Conv2D(128, kernel_size=(3, 3),
activation='relu'))
```

Let’s go to the summary of the model so that we can know how convolutional layers reduce the size of the input.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 22, 22, 128)       73856
=================================================================
Total params: 92,672
Trainable params: 92,672
Non-trainable params: 0
_________________________________________________________________
```

In the image which is a summary of the model which we have created, we can clearly see that with every convolutional layer the size output is changing in decreasing order. By padding, we can resolve this problem.

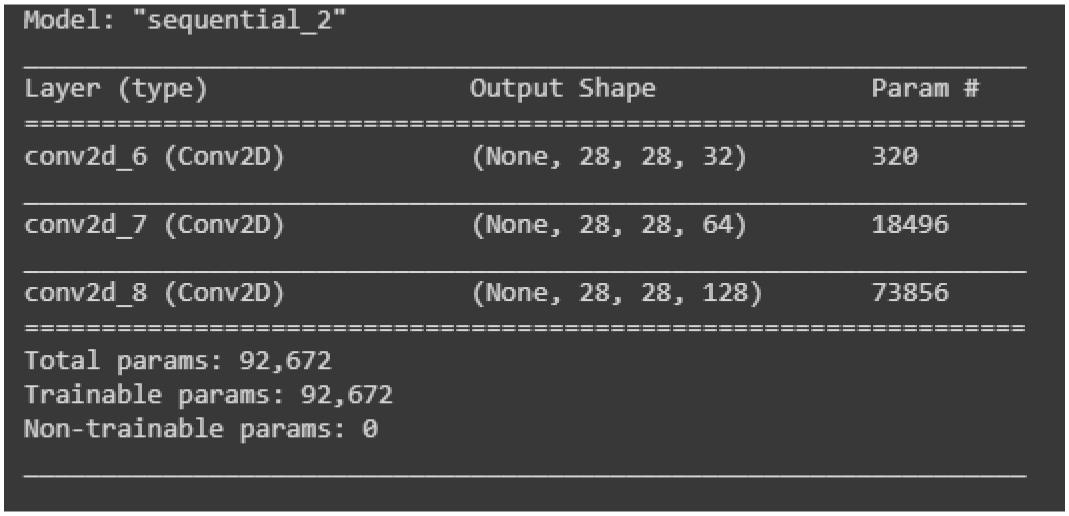
Same Padding

In this type of padding, the padding layers append zero values in the outer frame of the images or data so the filter we are using can cover the edge of the matrix and make the inference with them too.

Below the model is an example of how we can create a model with the same padding.

```
models = Sequential()
models.add(Conv2D(32, kernel_size=(3, 3),
activation='relu', input_shape=(28, 28, 1))),
padding='same'))
models.add(Conv2D(64, kernel_size=(3, 3),
activation='relu', padding='same'))
models.add(Conv2D(128, kernel_size=(3, 3),
activation='relu', padding='same'))
```

Let's check the summary.



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 64)        18496
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 128)       73856
=================================================================
Total params: 92,672
Trainable params: 92,672
Non-trainable params: 0
_________________________________________________________________
```

Here we can compare the summary of the model with the same padding and without padding. We can clearly see here the size of out is remaining the same. For each layer, the feature map dimensions are the same. This is how we can overcome the problem of reduction of output size. Every time the output shape is 28 * 28.

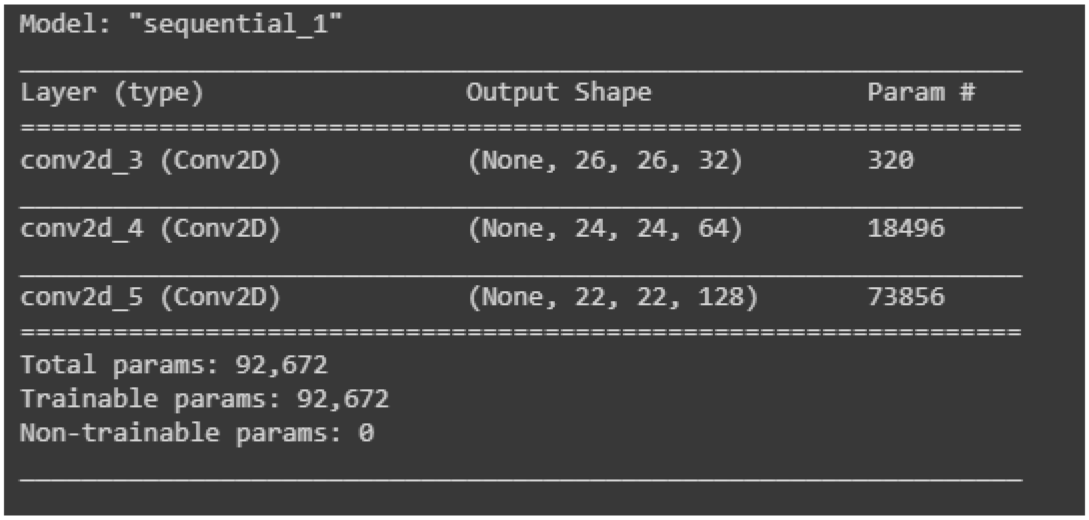
Valid Padding

This type of padding can be considered as no padding. Why is there no padding we will understand after the example of a model? Let's just look at the example.

```
modelv = Sequential()
modelv.add(Conv2D(32, kernel_size=(3, 3),
activation='relu', input_shape=input_shape, padding =
'valid'))
modelv.add(Conv2D(64, kernel_size=(3, 3),
activation='relu', padding = 'valid'))
modelv.add(Conv2D(128, kernel_size=(3, 3),
activation='relu', padding = 'valid'))
```

Here we have created a model with three two dimensional convolutional layers by adding valid padding on them.

Let's check the summary of the model.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 22, 22, 128)       73856
=================================================================
Total params: 92,672
Trainable params: 92,672
Non-trainable params: 0
_________________________________________________________________
```

Here we can see that there is no change in the summary in comparison to the summary of the model without padding. So here one question will arise in the reader's mind: why is it required?

So in this, we really don't apply padding but we assume that every pixel of the image is valid so that the input can get fully covered by the filter wherein a simple model assumes corners are invalid. And do not consider them in the coverage area.

Note- while using valid padding use them with max-pooling layers in the same way.

In the same padding, we use every point/pixel value while learning the model wherein the valid padding, we consider every point as valid so nothing can be left; it does not work with the size of the input, it works on validation of pixel value.

In VALID (i.e. no padding mode), Tensorflow will drop right and/or bottom cells if your filter and stride don't fully cover the input image. Where the same padding model tries to spread similar padding across the frame of the image.

Causal Padding

This is a special type of padding and basically works with the one-dimensional convolutional layers. We can use them majorly in time series analysis. Since a time series is sequential data it helps in adding zeros at the start of the data. Which also helps in predicting the values of early time steps.

We use this padding with convolutional layers which basically means every layer is using the learnt part of the previous layer and again the learnt part of the will go to the next layer and this is how it all works. So considering a time series if any forecasted value can be used in forecasting the next time step value it can become more helpful and accurate.

A model architecture consisting of causal padding with a kernel size of four can be represented as:

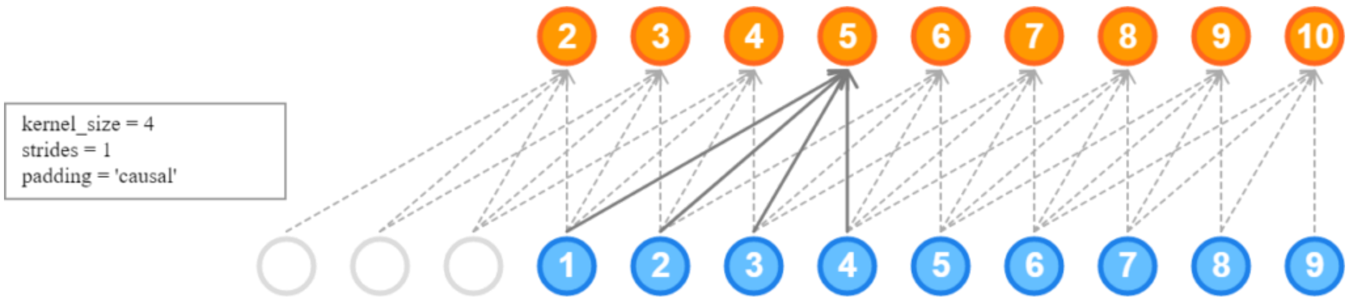
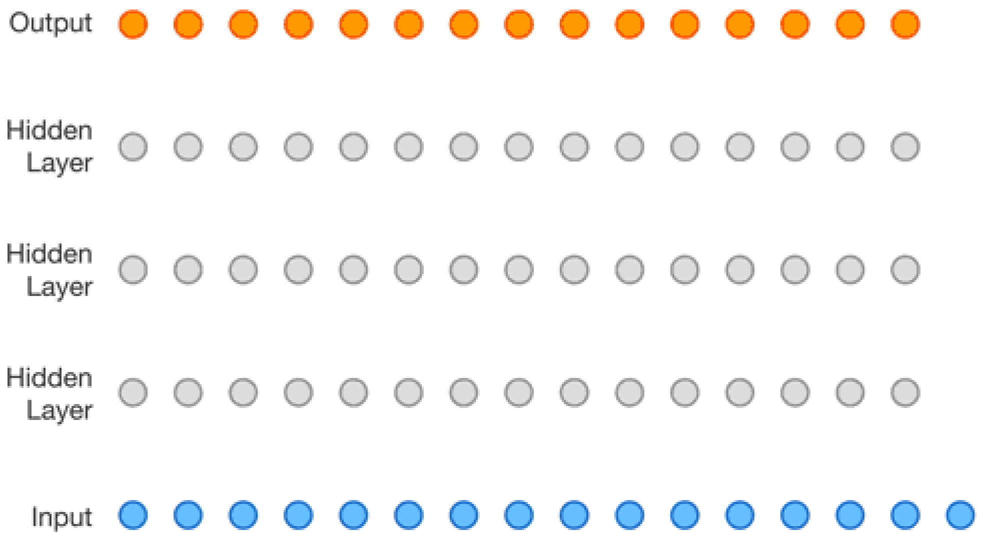


Image source (https://i.stack.imgur.com/NmYZJ.png)

And a whole time series model architecture with a fully connected layer and causal padding can be like:

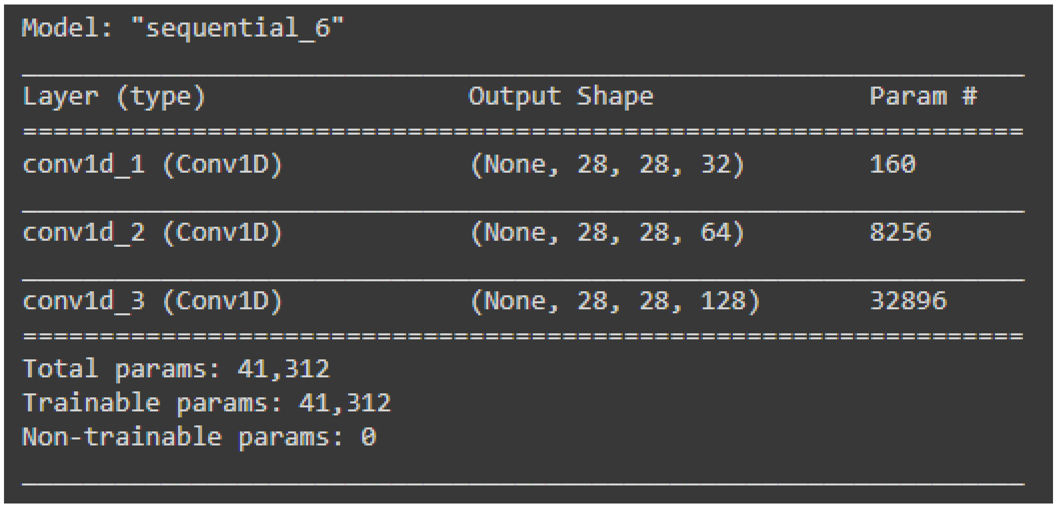


Let's see how we can implement this.

```
from keras.layers import Conv1D
modelc = Sequential()
modelc.add(Conv1D(32, kernel_size=4, activation='relu',
input_shape=(28, 28, 1), padding='causal'))
modelc.add(Conv1D(64, kernel_size=4, activation='relu',
padding='causal'))
modelc.add(Conv1D(128, kernel_size=4, activation='relu',
padding='causal'))
```

Here we have made a model with three one dimensional layers and causal padding.

Let's check the summary.



```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 28, 28, 32)        160
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 28, 28, 64)        8256
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 28, 28, 128)       32896
=================================================================
Total params: 41,312
Trainable params: 41,312
Non-trainable params: 0
_________________________________________________________________
```

Here we can see that the size of the output is not changing which means that it also helps by adding zero values but in the one-dimensional convolutional layer.

Final Words

Here in the article, we have seen how we can deal with the problem of the simple convolutional layer by using different padding. We have seen how these padding methods are different from each other. Where we can use the same and valid padding with two-dimensional convolutional layers and causal padding with one-dimensional convolutional layers.