$$O = \left(\frac{i - k + 2P}{S}\right) + 1 \qquad \boxed{\begin{matrix}P=0\\ S=1\end{matrix}}$$

=>

Considering an input image of size (28x 28 x 1)

model = . Sequential()

model. add (Conv2D (32, kernel_size = (33), activation = "relu",
input_shape = (28, 28, 1))

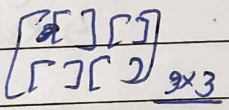
↳ Resultant size = (28 − 3) + 1 => (26×26×32)

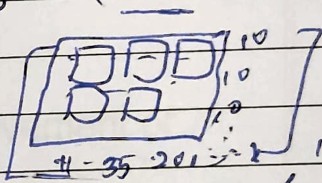
model. add (Max Pooling 2D ((2x2)))

↳ Resultant size = $\left\lfloor \frac{26}{2} \right\rfloor$ = (13 × 13 × 32)

model. add (Conv 2D (64, (3x3), activation = "relu")

↳ Resultant size = (13 − 3) + 1 = (11x11 x 64)





model. add (Max Pooling 2D ((2x2)))

Or Avg Pooling 2D

↳ Resultant size = $\left\lfloor \frac{11}{2} \right\rfloor$ = (5x5x64)

↓

Feature Map

Padding → TRX

Since the size of feature Maps obtained from this Layer is '11x11 x 64', & size of Maxpooling is (2x2), => This will lead to losses. Since the input size is not perfectly divisible by the Max pooling size.

We may use 2 types of kernels of sizes (4X4) or (2x2) in order to get the feature map of an even size so that possibility of any feature getting lost is nullified.

For (4x4) kernel size :-

model. add (Conv2D (64, (4x4), activation = 'relu')

=> 13 − 4 + 1 => (10 x 10 x 64)

For (2x2) kernel / filter size :-

model.add(Conv2D(64, (2x2), activation = 'relu')
↳ Resultant Size = 13 - 2 + 1 => (12x12x64)

After (4x4) kernel usage :-

=> model.add(MaxPooling2D((2x2))
) -> Resultant size
= $\lfloor \frac{10}{2} \rfloor$ => (5x5x64)

model.add(Flatten())
↳ Size = (1600,)

model.add(Dense(64,
activation='relu'))
↳ Size =

Model.add(Dense(10))

After (2x2) kernel usage :-

model.add(MaxPooling2D((2x2)))
↳ Resultant Size
= $\lfloor \frac{12}{2} \rfloor$ = (6x6x64)

model.add(Flatten())
↳ Size = (2304,)

Model.add(Dense(64,
activation='relu'))

model.add(Dense(10))

Here, kernel of (2x2) size, when used in Conv2D is of much better use since when flattened, we are able to achieve more number of features (2304,) ~~the very problems~~ which may lead to better features.