EED380

APPLIED DEEP LEARNING

# DIGIT RECOGNIZER USING CONVOLUTIONAL NEURAL NETWORKS

By :

Ishdeep Singh Chadha

1810110087

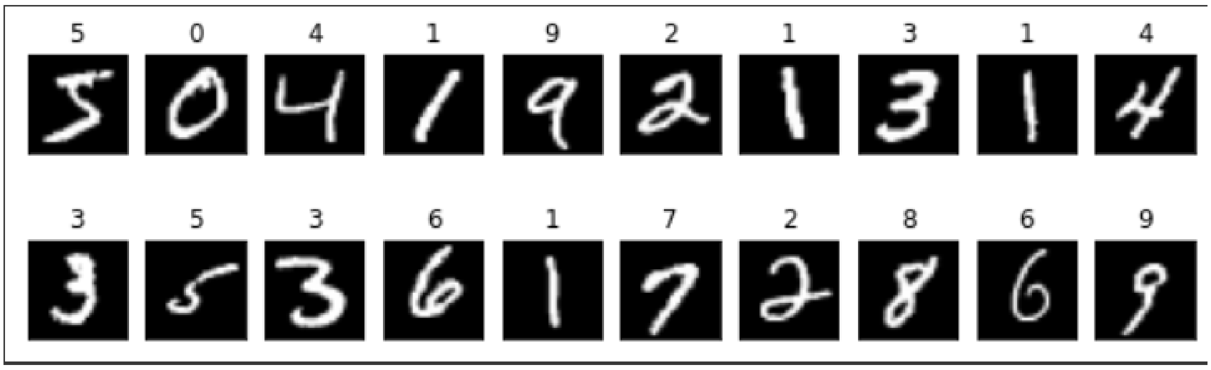
## Introduction

Deep Learning is a machine learning method that trains computers to do what easily falls into place for people: learning through examples. With the utilization of deep learning methods, human attempts can be diminished in perceiving, learning, recognizing and in a lot more regions. Using deep learning, the computer learns to carry out classification works from pictures or contents from any document. Deep Learning models can accomplish state-of-art accuracy, beyond the human level performance.

The aim of this project is to implement a classification algorithm to recognize the handwritten digits with Deep Learning calculation like multilayer CNN utilizing Keras and Tensorflow. TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

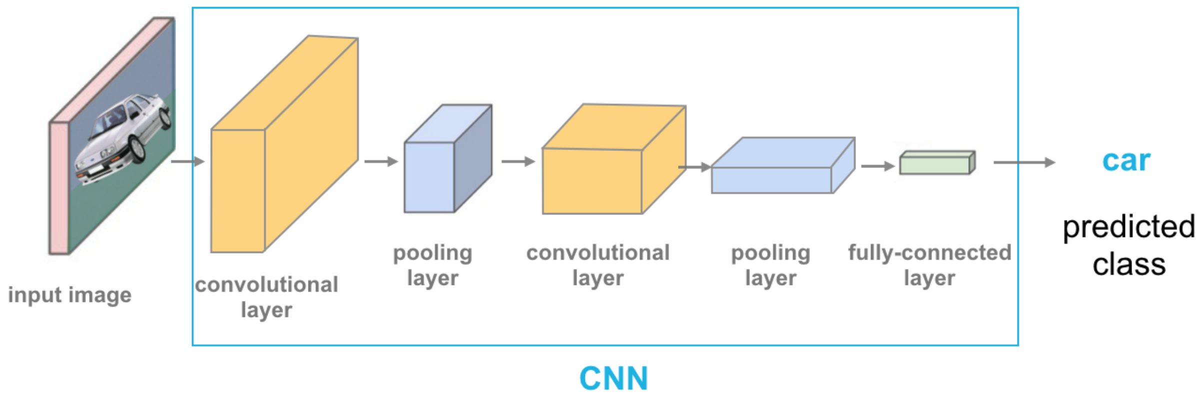
## The MNIST dataset

This is probably one of the most popular datasets among machine learning and deep learning enthusiasts. The MNIST dataset contains 60,000 training images of handwritten digits from zero to nine and 10,000 images for testing. So, the MNIST dataset has 10 different classes. The handwritten digits images are represented as a 28×28 matrix where each cell contains grayscale pixel value. Each pixel has a single pixel-value associated with it , indicating the lightness or darkness of that pixel , with higher numbers meaning darker. This pixel-value is an integer between 0 and 255.



# Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The term 'Convolution" in CNN denotes the mathematical function of convolution which is a special kind of linear operation wherein two functions are multiplied to produce a third function which expresses how the shape of one function is modified by the other. In simple terms, two images which can be represented as matrices are multiplied to give an output that is used to extract features from the image**.** The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.



## Architecture of a Convolutional Neural Network :

### 1. Convolutional Layer :

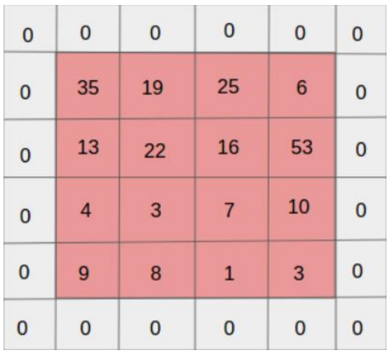
This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a kernel of a particular size kxk. By sliding the kernel over the input image, the dot product is taken between the kernel and the parts of the input image with respect to the size of the kernel (kxk). The output is termed as the Feature map(after adding bias) which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image. There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying **Valid Convolution** in case of the former, or **Same Convolution** in the case of the latter.

## General Formula for output size of convolution layer :

## Output size = [(n + 2p – k) / s + 1 X ( n + 2p – k) / s + 1 ]

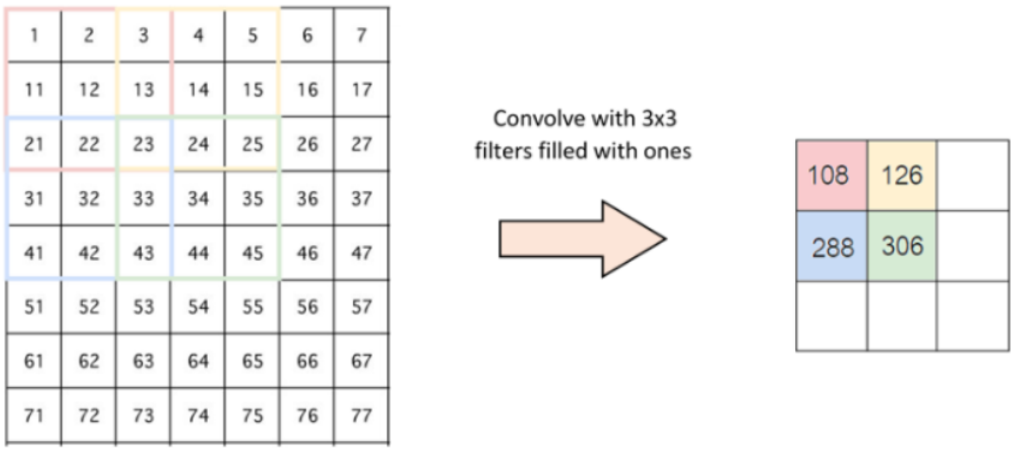
Where n = size of input image , p = zero padding , s = strides

➔ Zero Padding = Zero-padding refers to the process of symmetrically adding zeroes to the input matrix. It's a commonly used modification that allows the size of the input to be adjusted to our requirement. It is mostly used in designing the CNN layers when the dimensions of the input volume need to be preserved in the output volume.



| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 35 | 19 | 25 | 6 | 0 |
| 0 | 13 | 22 | 16 | 53 | 0 |
| 0 | 4 | 3 | 7 | 10 | 0 |
| 0 | 9 | 8 | 1 | 3 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

➔ Strides = Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on.



| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 |

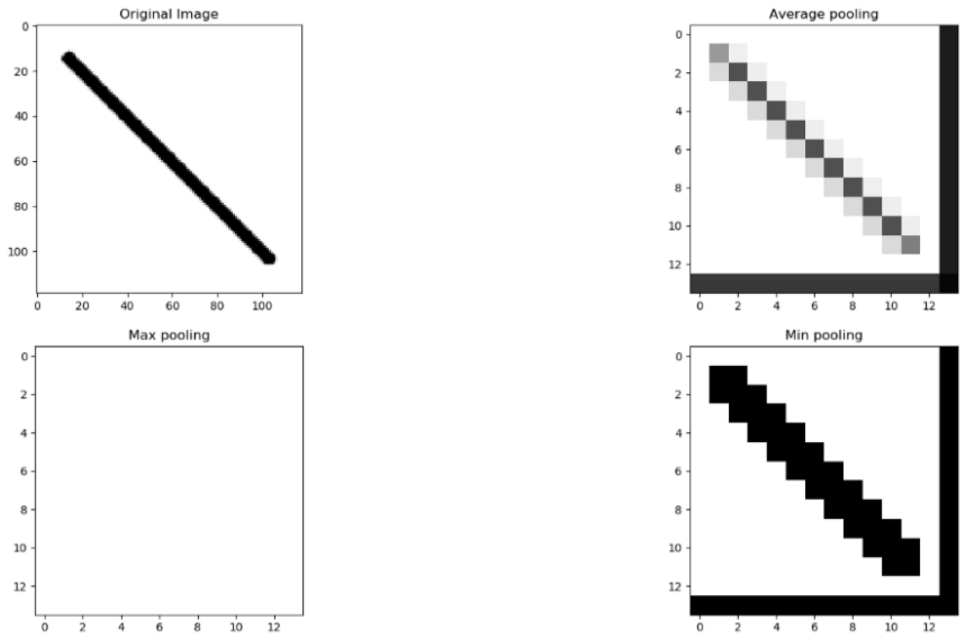
| | | |
|---|---|---|
| 108 | 126 | |
| 288 | 306 | |
| | | |

## 2. Pooling Layer

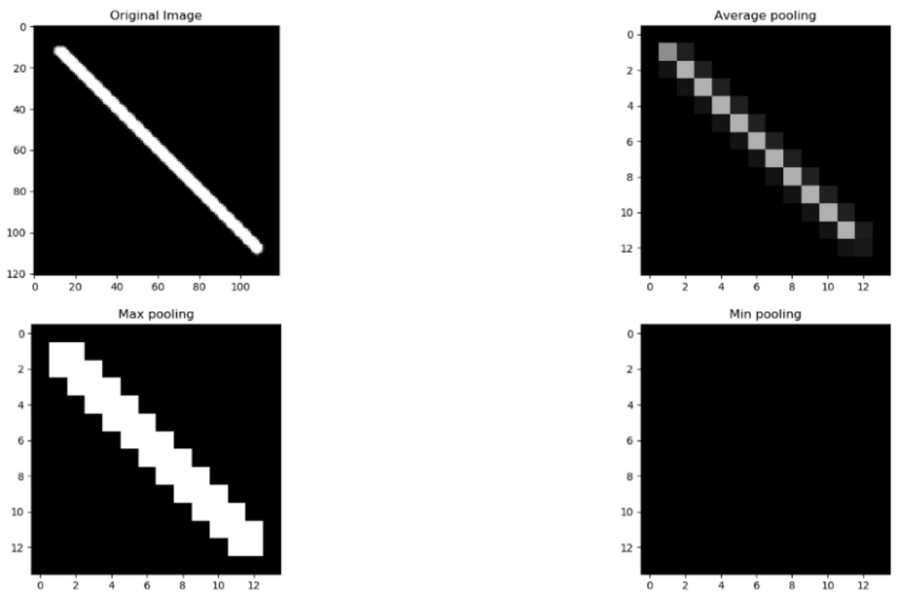
The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs and avoid overfitting. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

The three types of pooling operations are:

1. Max pooling: The maximum pixel value of the batch is selected.
2. Min pooling: The minimum pixel value of the batch is selected.
3. Average pooling: The average value of all the pixels in the batch is selected.



Min pooling gives better result for images with white background and black object

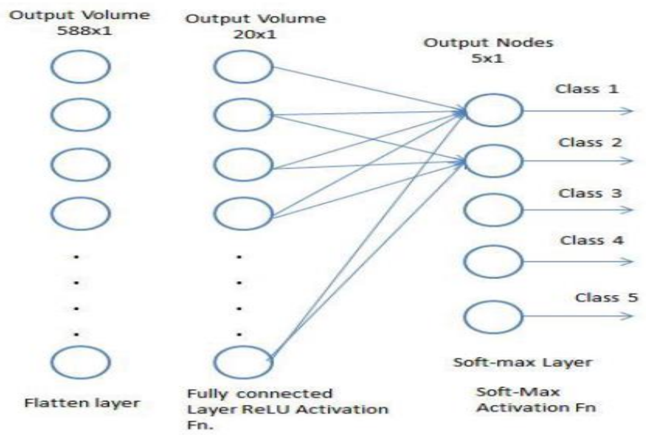


Max pooling gives better result for the images with black background and white object (Ex: MNIST dataset)

As the images of digits in our dataset have a black background we have used maxpooling.

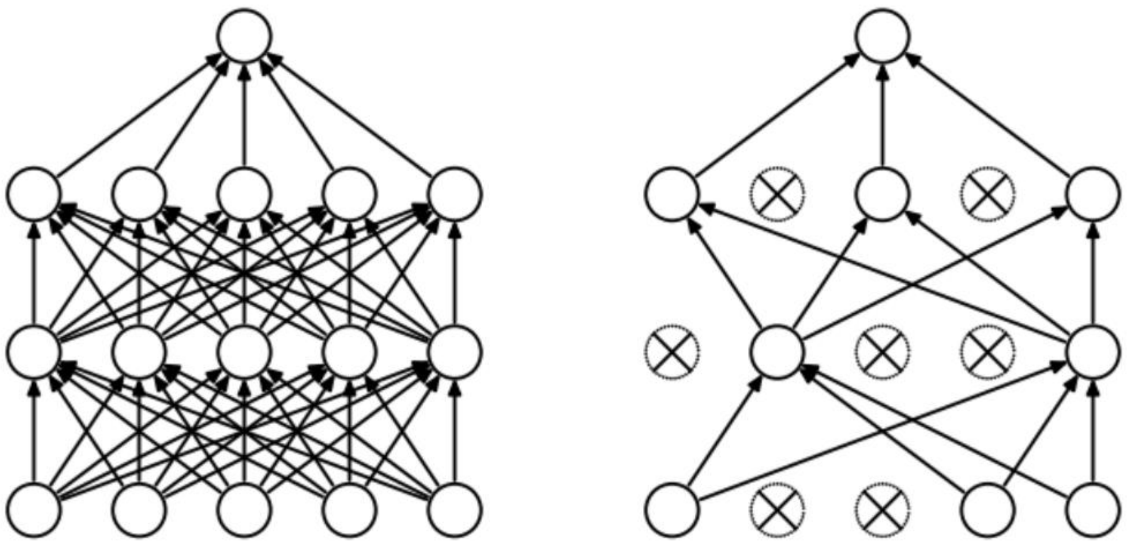
## 3. Fully Connected Layer

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.
In this, the input image from the previous layers is flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place.



## 4.Dropout and Batch Normalization

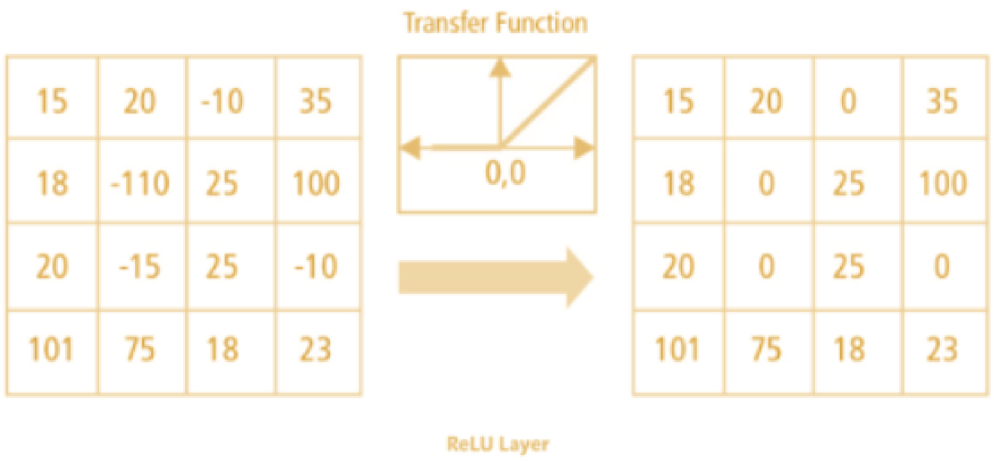
Dropouts are the regularization technique that is used to prevent overfitting in the model. Dropouts are added to randomly switching some percentage of neurons of the network. When the neurons are switched off the incoming and outgoing connection to those neurons is also switched off. This is done to enhance the learning of the model.

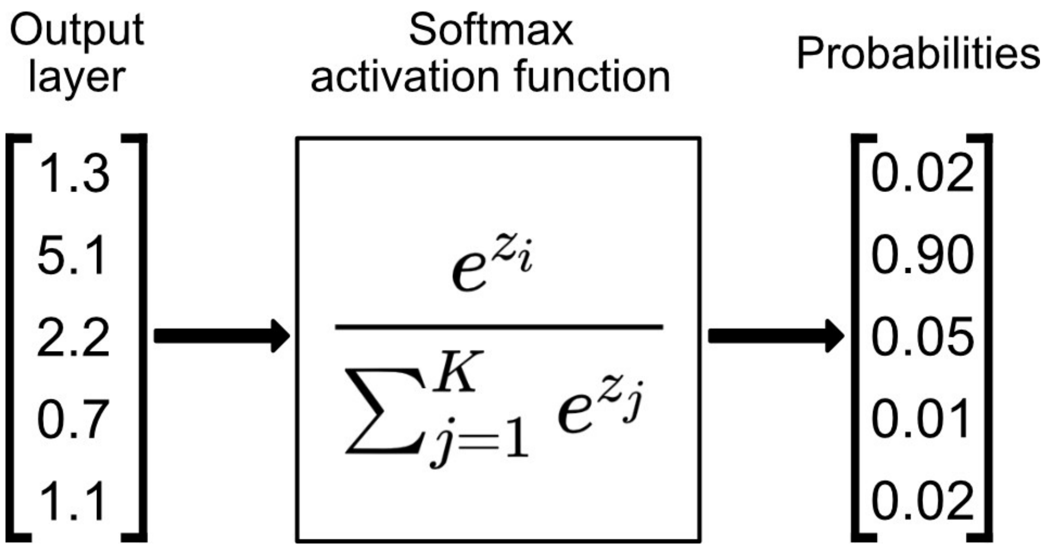

(a) Standard Neural Net (b) After applying dropout.

Batch normalization is a layer that allows every layer of the network to do learning more independently. It is used to normalize the output of the previous layers. The activations scale the input layer in normalization. Using batch normalization learning becomes efficient also it can be used as regularization to avoid overfitting of the model. The layer is added to the sequential model to standardize the input or the outputs.

## 5. Activation Functions

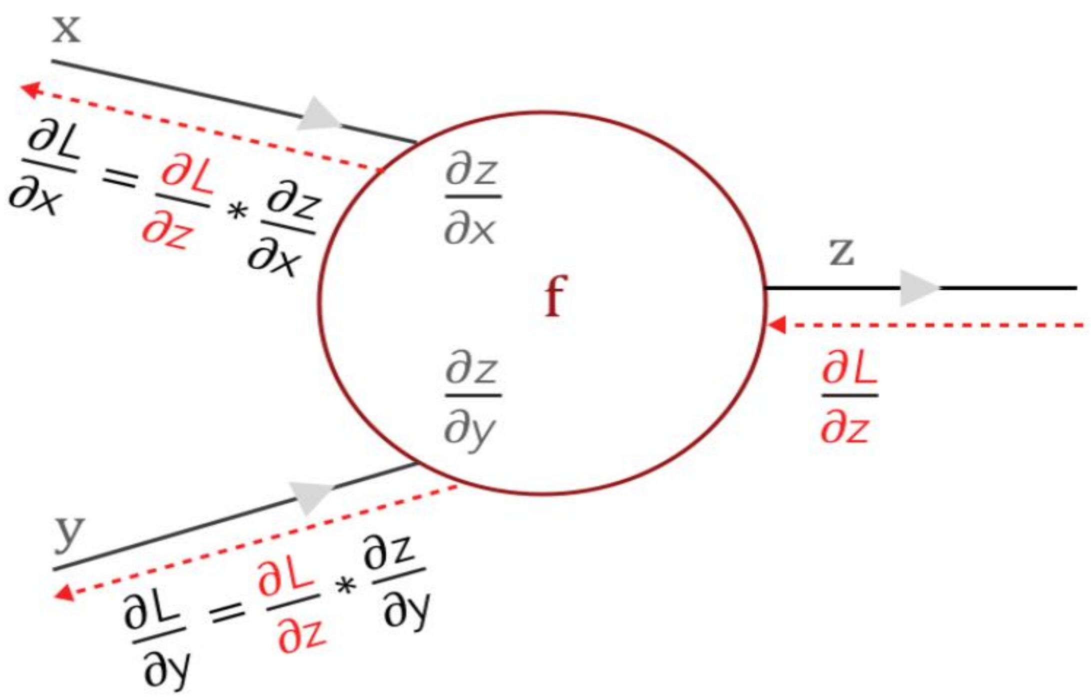
Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. For this model , we have used ReLU and softmax activation functions.




Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

## Forward and Backward Propagation in CNN

For the forward pass, we move across the CNN, moving through its layers and at the end obtain the loss, using the loss function. And when we start to work the loss backwards, layer across layer, we get the gradient of the loss from the previous layer as ∂L/∂z. In order for the loss to be propagated to the other gates, we need to find ∂L/∂x and ∂L/∂y.



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial y}$

$\frac{\partial z}{\partial x}$ & $\frac{\partial z}{\partial y}$ *are local gradients*

$\frac{\partial L}{\partial z}$ *is the loss from the previous layer which has to be backpropagated to other layers*

## Building our CNN model

- Shape of our data :

```
Shape of X_train:  (60000, 28, 28)                    Shape of y_train:  (60000,)

Shape of X_test:  (10000, 28, 28)                      Shape of y_test:  (10000,)
```

Each image have size of 28*28 pixels and hence (28,28,1) , 1 in last part is to specify color depth of the pixel. Each pixel has a single pixel-value associated with it , indicating the lightness or darkness of that pixel , with higher numbers meaning darker. This pixel value is an integer between 0 and 255. To improve the performance of our model we can normalize these pixel values to have range between 0 to 1.

We will use a Sequential model which refers to a linear stack of layers. It can be first initialized and then we add layers using add method. The layers added are as follows :

1. Conv2D is a 2D Convolutional layer which performs convolution between our input image matrix and weight matrix. The parameters used are :
   (i) Filters – the number of filters(kernels) used with this layer.
   (ii) Kernel_size - the dimension of the kernel. It can only take a odd value.
   (iii) Activation – is the activation function used. For ex : ReLU , softmax.
   (iv) Kernel_intializer – the function used for initializing the weights inside the kernel matrix. We have used he_normal initialization. It draws samples from a truncated normal distribution centered on 0 with stddev = sqrt(2 / fan_in) where fan_in is the number of input units in the weight tensor.
   (v) Input_shape – is the shape of the image presented to the CNN , in our case is 28*28*1.
   (vi) Value of strides and zero padding.

2. BatchNormalization
3. MaxPooling2D : Parameters – pool_size representing the factors by which to downscale in both directions. Ex – (2,2)
4. Dropout : We have used two different values of dropout of 0.2 and 0.3.
5. Flatten is use to convert the final feature maps into a single 1D vector. This flattening step is needed so that you can make use of fully connected layers after some convolution layers. It combines all the found local features of the previous convolutional layers.
6. Dense is the final layers with one dense layer having units as equal to our total number of classes.

## Training our CNN model

### Optimizer used :

ADAM. Parameters are as follows :

- alpha. Also referred to as the learning rate or step size. The proportion that weights are updated (e.g. 0.001). Larger values (e.g. 0.3) results in faster initial learning before the rate is updated. Smaller values (e.g. 1.0E-5) slow learning right down during training
- beta1. The exponential decay rate for the first moment estimates (e.g. 0.9).
- beta2. The exponential decay rate for the second-moment estimates (e.g. 0.999). This value should be set close to 1.0 on problems with a sparse gradient (e.g. NLP and computer vision problems).
- epsilon. Is a very small number to prevent any division by zero in the implementation (e.g. 10E-8).

### Loss calculated while compiling our model : sparse_categorical_crossentropy.

In most cases CNNs use a cross-entropy loss on the one-hot encoded output. For a single image the cross entropy loss looks like this:

$$-\sum_{c=1}^{M}(y_c \cdot \log \hat{y}_c)$$

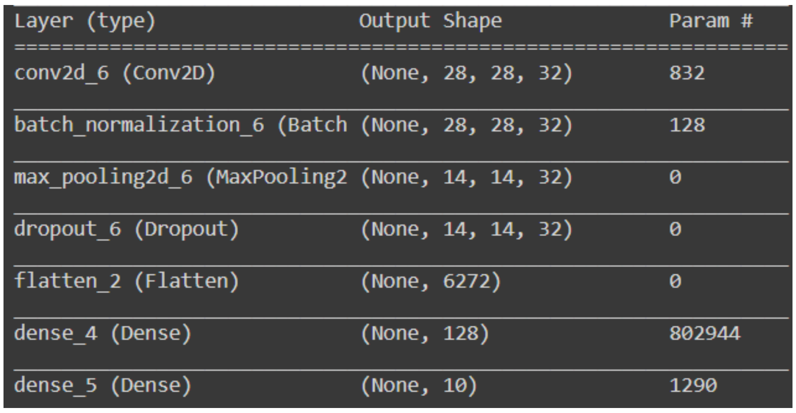
where M is the number of classes and y^cy^c is the model's prediction for that class (i.e. the output of the softmax for class c). Due to the fact that the labels are one-hot encoded and y is a vector of ones and zeroes, yc is either 1 or 0. Thus, out of the whole sum only one term will actually be added: the one with yc=1.

**Fitting our model :**

Number of epochs taken = 30.
Batch_size = 128
Validation_split = 0.2 (20% data from training set to be used for validation)

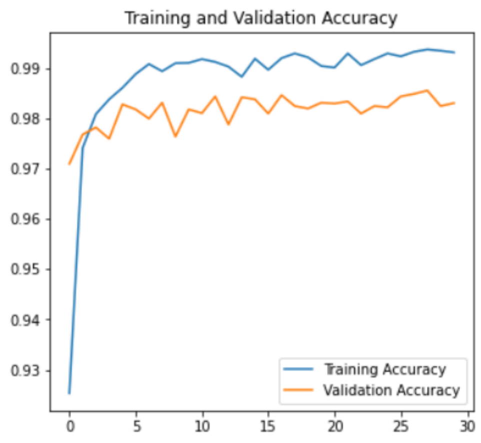
## Observations

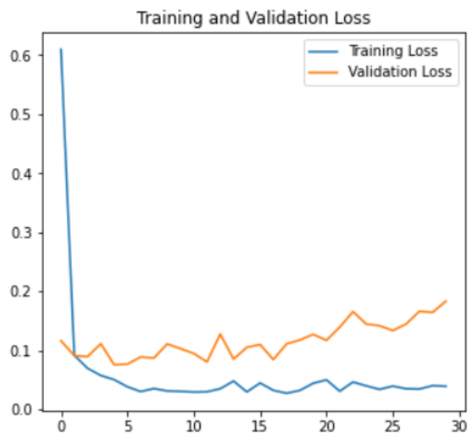
1. One Convolution layer with kernel size = (5,5) and stride = 1 and zero padding applied.
   Model summary :



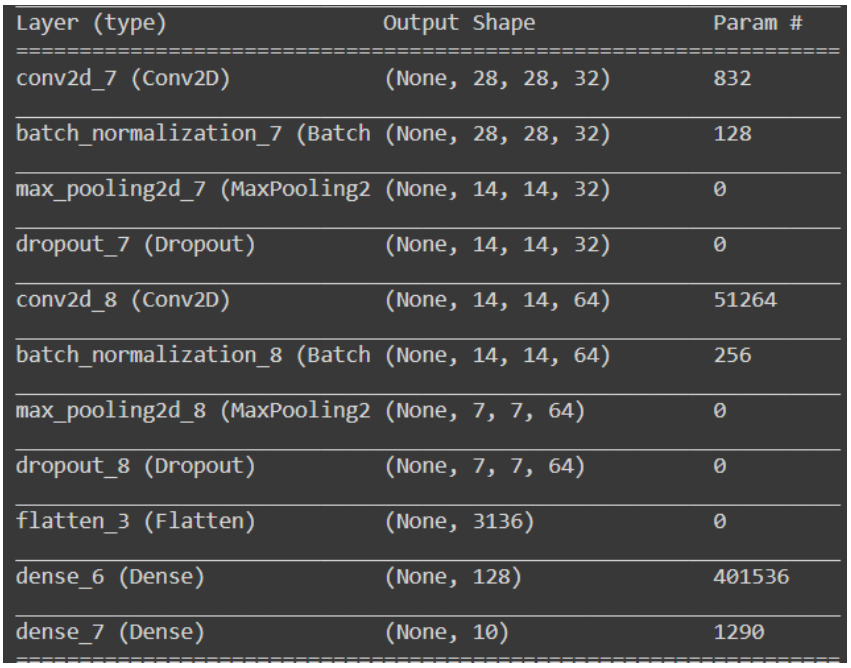
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
batch_normalization_6 (Batch (None, 28, 28, 32)        128
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_6 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               802944
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
```

   Test Accuracy = 98.39 %
   Test Loss = 0.143



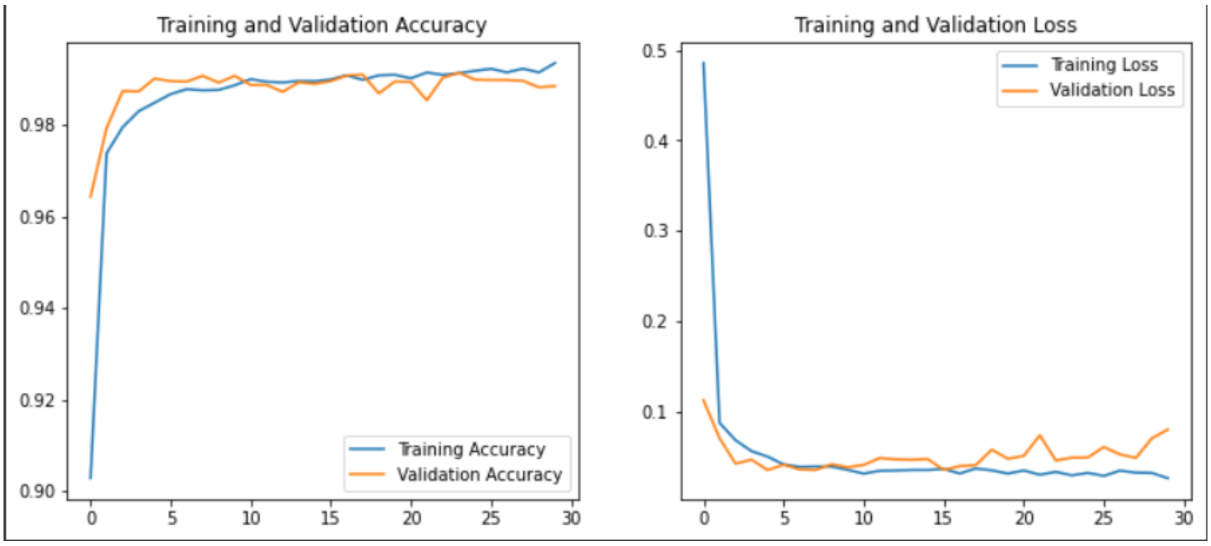


2. Two Convolution layers with kernel size = (5,5) and stride = 1 and zero padding applied.

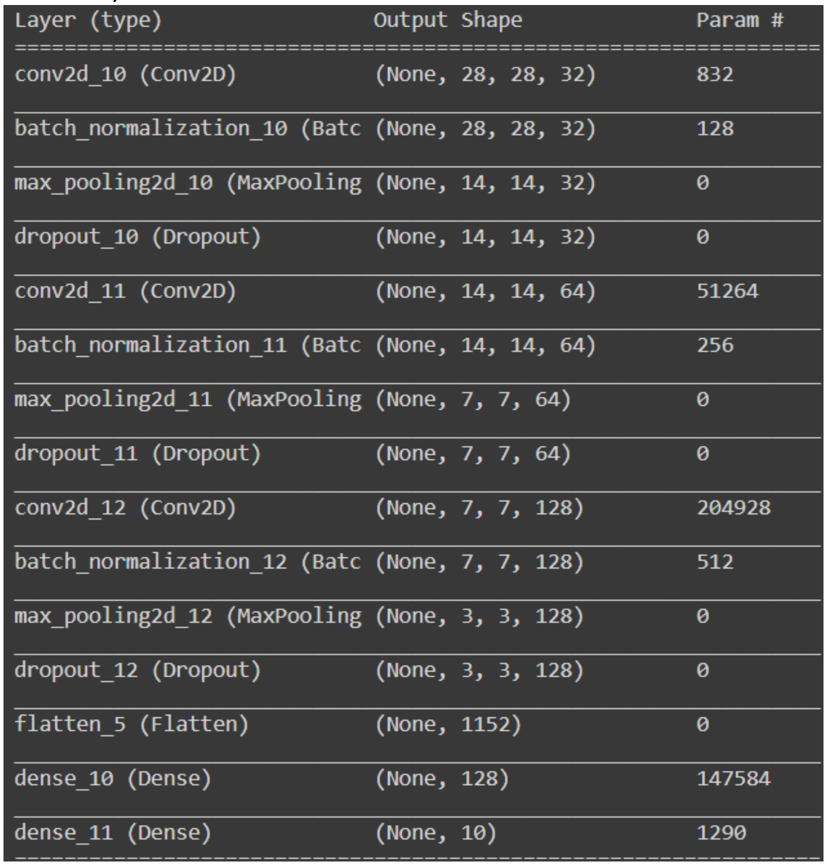


```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 28, 28, 32)        832

batch_normalization_7 (Batch (None, 28, 28, 32)        128

max_pooling2d_7 (MaxPooling2 (None, 14, 14, 32)        0

dropout_7 (Dropout)          (None, 14, 14, 32)        0

conv2d_8 (Conv2D)            (None, 14, 14, 64)        51264

batch_normalization_8 (Batch (None, 14, 14, 64)        256

max_pooling2d_8 (MaxPooling2 (None, 7, 7, 64)          0

dropout_8 (Dropout)          (None, 7, 7, 64)          0

flatten_3 (Flatten)          (None, 3136)              0

dense_6 (Dense)              (None, 128)               401536

dense_7 (Dense)              (None, 10)                1290
=================================================================
```

Test Accuracy : 99.11 %
Test Loss : 0.048

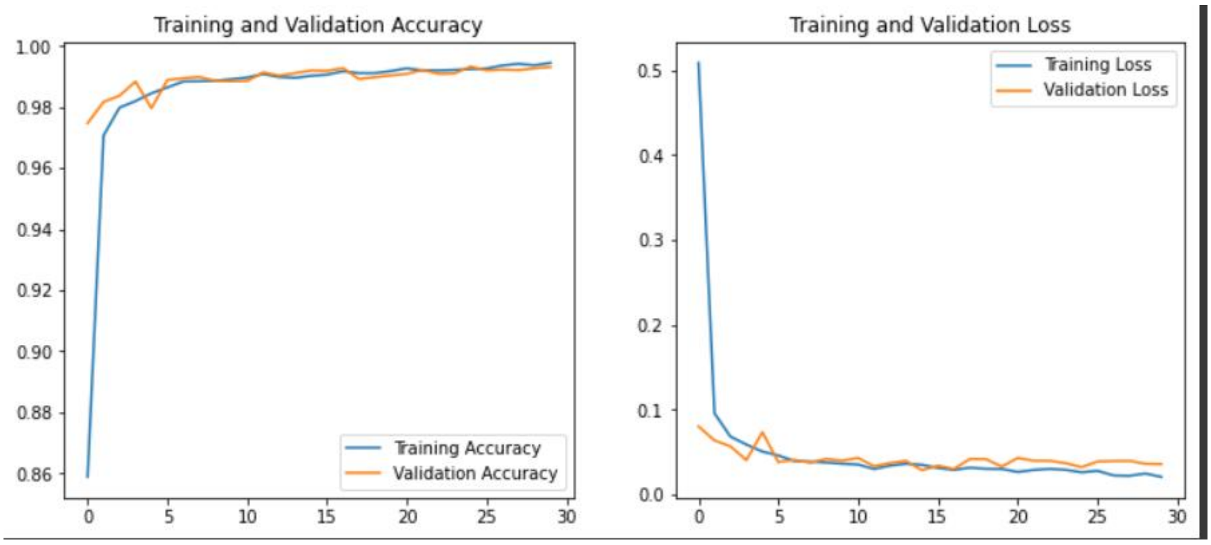


3. Three Convolution layers with kernel size = (5,5) and stride = 1 and zero padding applied.
   Model Summary :



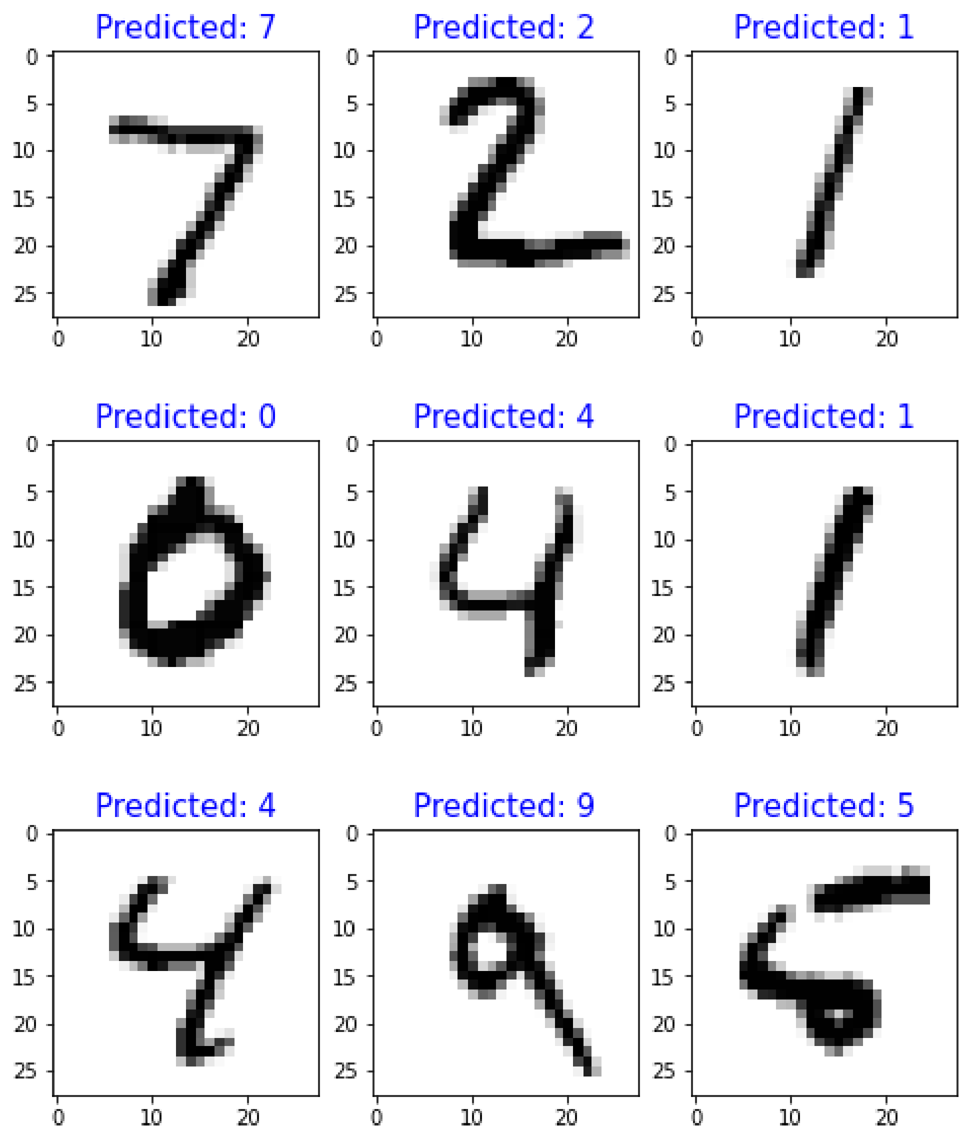
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
batch_normalization_10 (Batc (None, 28, 28, 32)        128
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_10 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 64)        51264
_________________________________________________________________
batch_normalization_11 (Batc (None, 14, 14, 64)        256
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_11 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 7, 7, 128)         204928
_________________________________________________________________
batch_normalization_12 (Batc (None, 7, 7, 128)         512
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 3, 3, 128)         0
_________________________________________________________________
dropout_12 (Dropout)         (None, 3, 3, 128)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               147584
_________________________________________________________________
dense_11 (Dense)             (None, 10)                1290
_________________________________________________________________
```

Test Accuracy : 99.32%
Test Loss : 0.023



Out of the three experiments done we can see that the model with three convolution layers give the best accuracy and the lowest test loss.
Also we can see that even after 30 epochs the validation accuracy does not shoot up at any point and saturates with the training accuracy thus we can say that the model is not overfitting.
Also Running the same model with kernel size = (3,3) gave an accuracy of 99.11% therefore we can go ahead with our original kernel size.

## Predictions and Results



We can see that our model has correctly predicted all of the above digits.

## Conclusion

In this report we have seen how we can use a convolution neural network to predict handwritten digits and what all parameters are required in order to do so. For future , this technique can be used to many other applications which not only classify images but also text or speech. Also , CNN models can also be used to not only classify grayscale images but also coloured images having RGB values. The only difference it would have is that the depth of each image would be 3 instead of 1.