Lenet

In our last session we discussed about TFOD and TFOD is a kind of framework using for **Transfer learning.**

In transfer learning is been done based on a base model.

So what is the actual architecture behind the transfer learning?

Today we are going to start our discussion architecture.

Previously we discussed CCN, Convolution, Pooling, Padding, Kernel size (Filters).

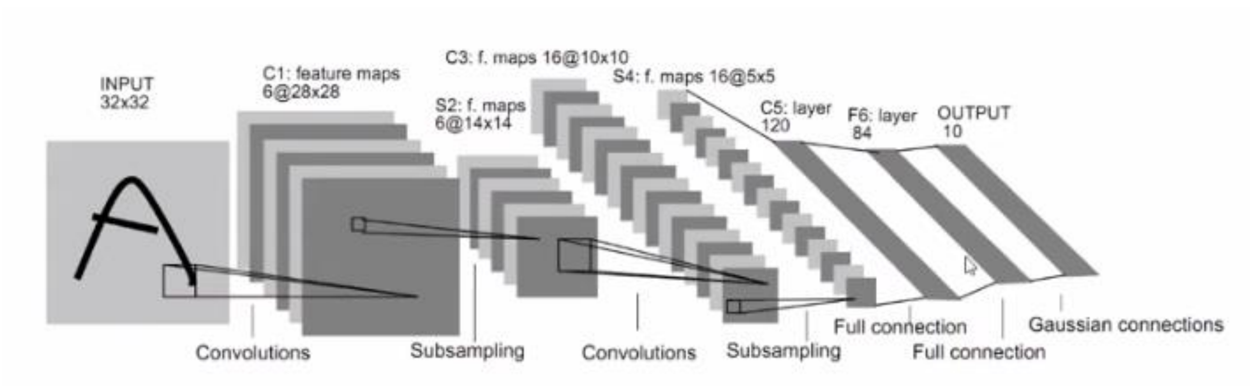
LeNet:

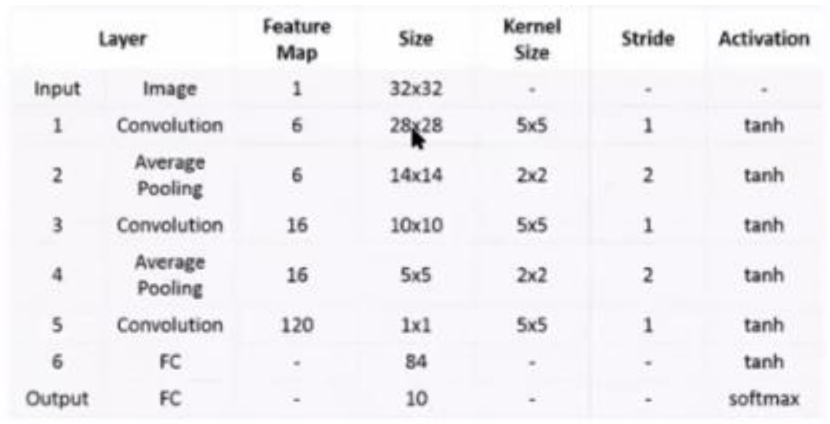
This architecture was developed in the year 1998 by
Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

This was given as a breakthrough in terms of detecting and recognizing in kind of an objects.

LeNet is one of the first and best model that time not now.

Here they have used one dataset called as mnist data set. Which is basically a collection of hand written digits.





Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

In this case they have tried to take black and white dataset of size 32X32 in LeNet.

On top of that what they have done is they have implemented a 6 kernels or 6 features and the finally will get of size 28X28

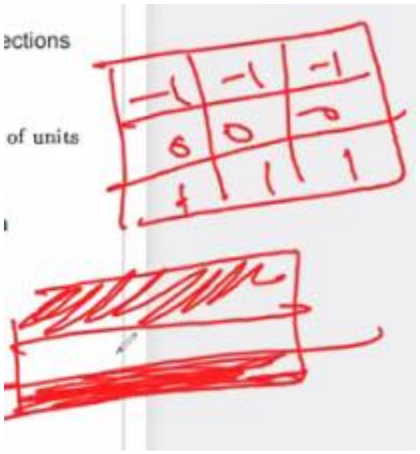
Here the Kernels is nothing but the property that we are trying to extract from the images.

As we discussed in our CNN, Whatever images you are going to take, every images is the combination of different colors, densities, sizes and shapes (circular, rectangular, square).

If you will be able to separate all of those things somehow then for sure you will be able to understand the pattern. Example human beings.

In our basic CNN model we have taken our Kernel size of 3X3.

Suppose I we are talk about this kind of a future.



So different kind of patterns are represented in the form of different kinds of matrices. This is called as Feature or Kernel.

As we discussed in our CNN architecture like how will be able to implement different

Kernels on top of the images and then how will be able to extract a data and that is called as a Convolution.

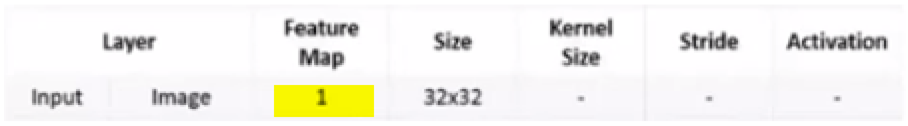
Then we talk about the Pooling (2X2) and out of that then I will try to extract maximum values which means more significant data in Maxpooling.

Padding – adding zero’s across the images.

Flattening operation – Whatever data you will be gathered and make into a single array and we need to pass those data’s into a Feed forward network and it will try to learn it inside a dense layer with the help of back propagation and also with the help of different kinds of loss and optimizers and then it will be able to recognize it.

In case of a CNN there are two kinds of learning are happening. Feature and in the second one it tries to understand a relationship between all those extracted features. So that it will be able to understand the particular object.

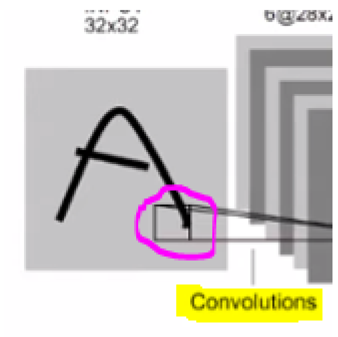
In this LeNet we have taken input of image size 32X32 and it is a grayscale image so that's a reason feature map is 1 which means it taking all the pixels.

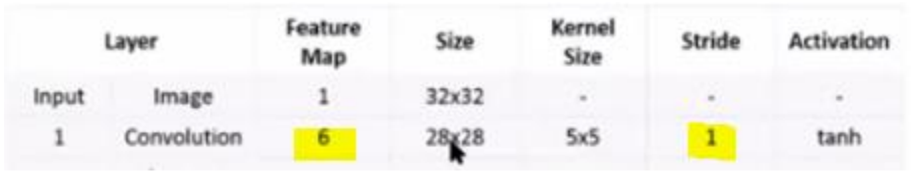


Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-

Step 1:

After that we are trying to do a Convolution



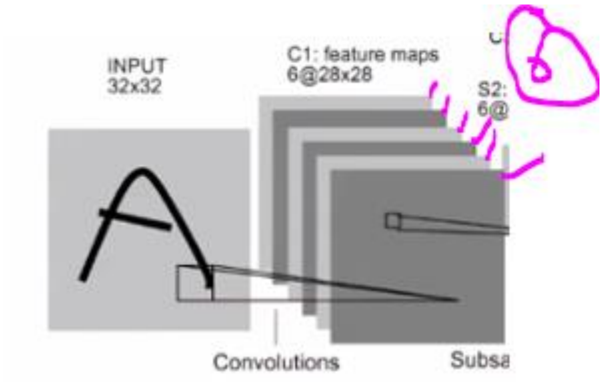


Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh

Which means we have created a kernel and we trying to move all those kernels with a stride = 1 and we apply the filter size of 5X5.

What is stride? – The no of shift horizontally and vertically, so that you will be able to cover the entire pixels.

And the number of feature is that we have implemented is 6. So that's the reason you will be able to see six boxes over here.

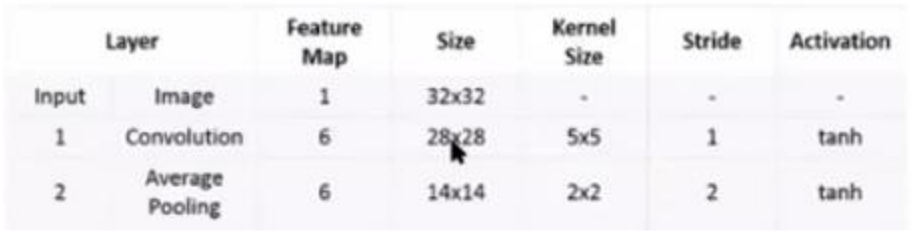


Which means every feature will apply on top of the original images. And try to extract the properties one after another in the form of matrices.

We have 6 filters and finally it gives six layers and every array has the size of 28X28

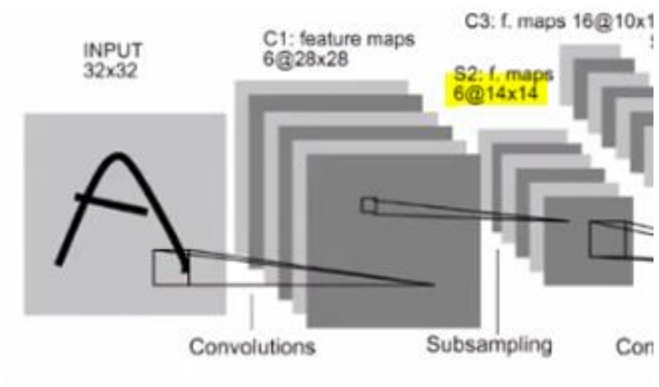
Step 2:

After applying this convolution (C1) over here then we go ahead with the Average pooling.



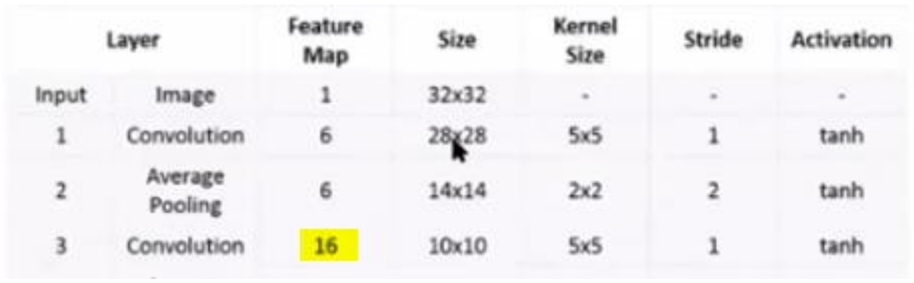
Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh

On every layer (6) we are implementing the average pooling. So finally we end up generating another layer of size 14X14 (6 layers) of kernel size 2X2 which means the feature size that we have taken along with a stride is 2 which means there is no overlap.

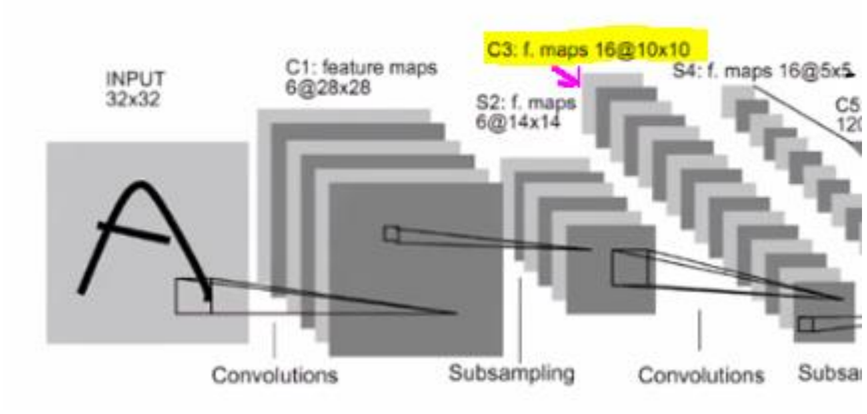


Step 3:

After average pooling again we are doing a convolution, this time we are trying to implement the feature of 16 and finally we are getting 10X10 of 16 layers or matrices. Here the kernel size we have used is 5X5 with stride 1.



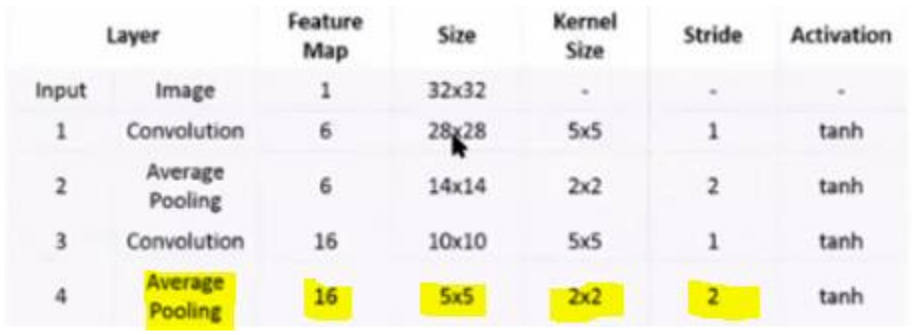
Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh



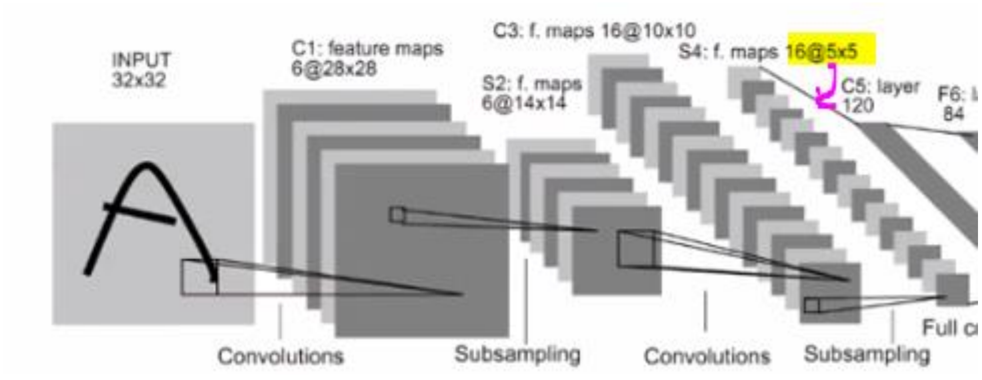
Step 4:

After this again we are calling Average pooling of strides 2 of kernel size 2X2

And finally you will be able to get 16 layer of properties or features of size 5X5.

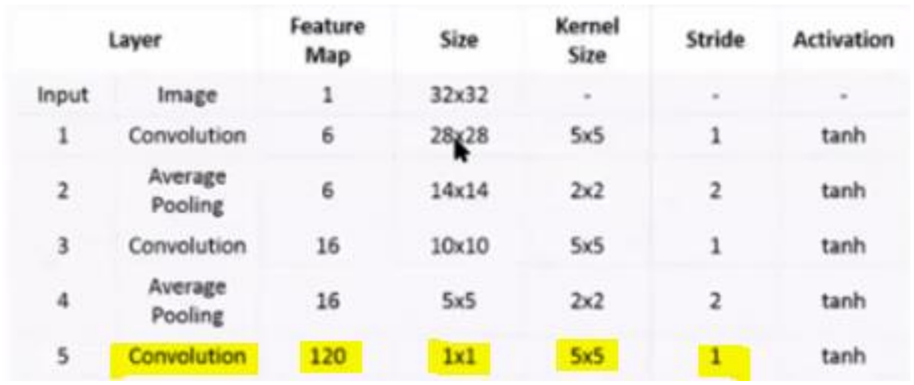


Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh



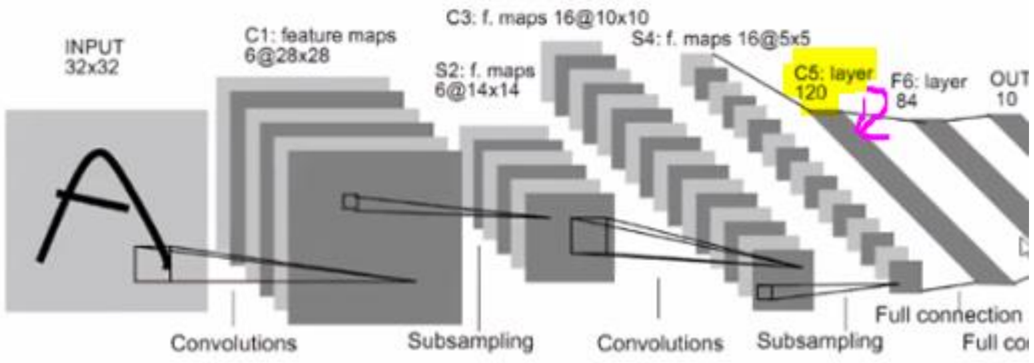
Once you will be able to get all of those things then again we are trying to do a convolution.

Step 5:



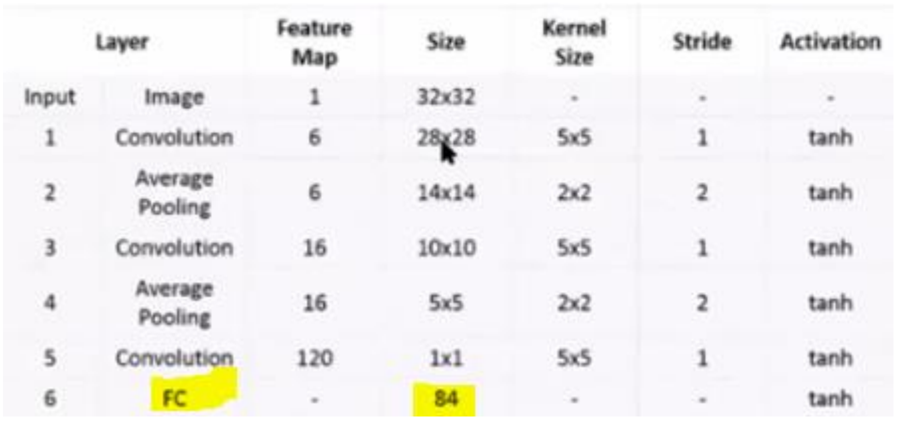
Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh

Here we were trying to consider basically a feature of 120 by applying the filter size 5X5 with stride 1 and we finally we are end up of creating a matrices of size 1X1 of single dimensional array (Flatten array).



Step 6:

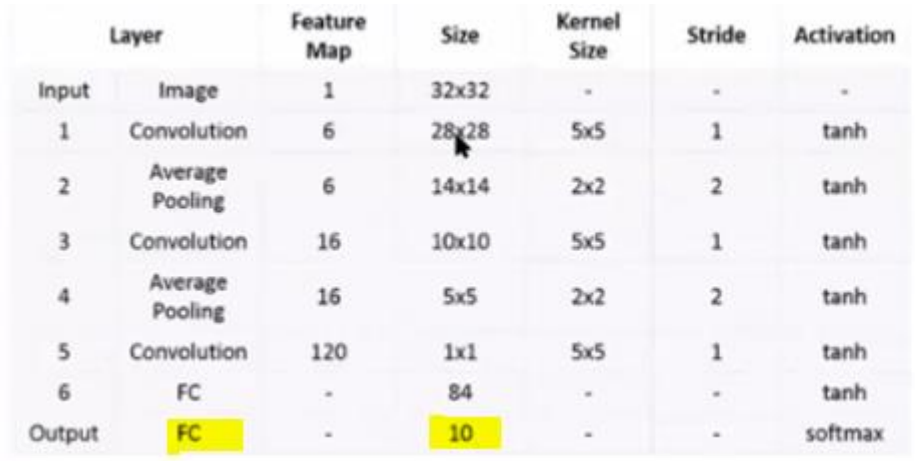
Now we are going to talk about a fully connected layer. In this we have 84 Neurons (Hidden layer)



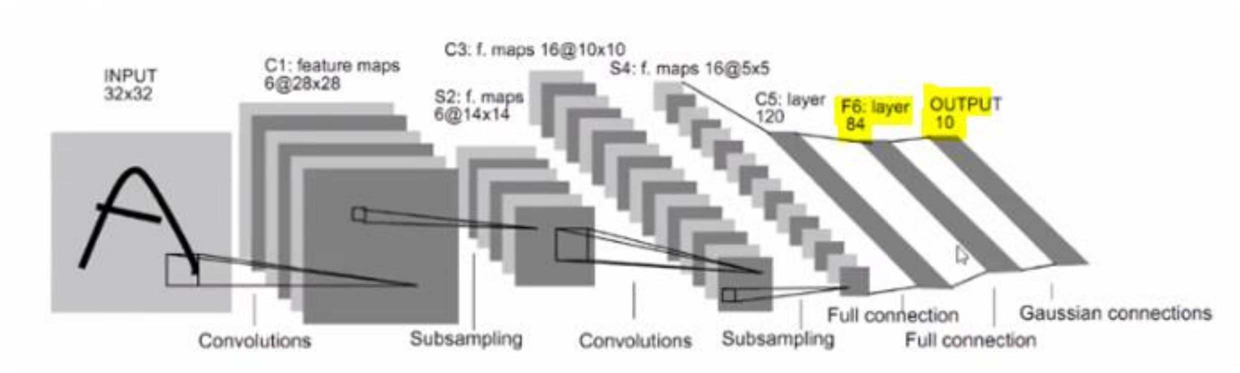
Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh

Step 7:

In a final layer we have 10 Neurons and which is called as output layer.

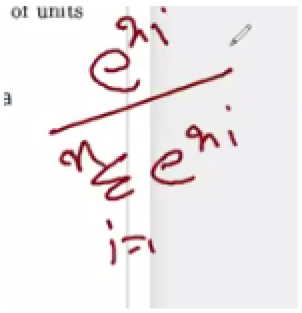


Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

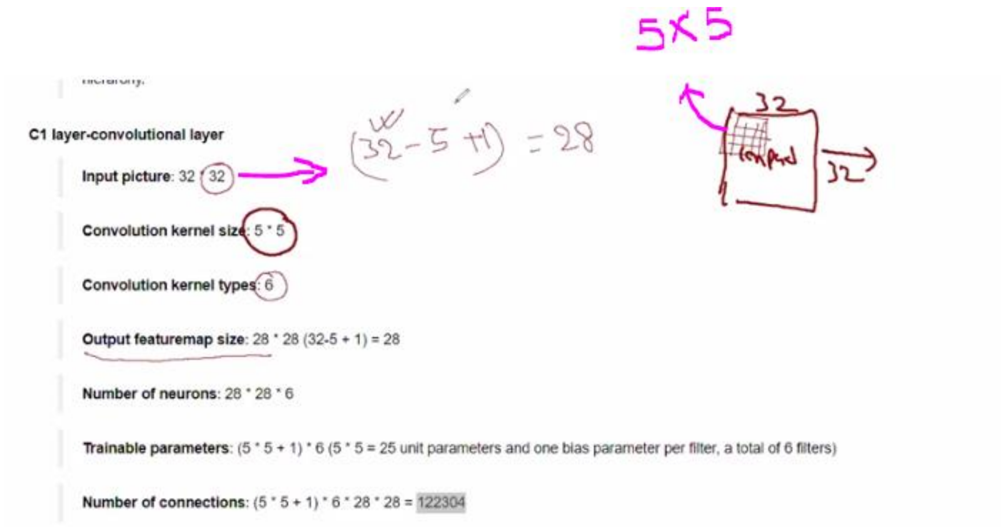


If I talk about the activation functions, everywhere we are using tanh functions and in the output layer we have used softmax activation function.

Softmax activation function will always try to find out the probability of one class among other classes.


$$\frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_i}}$$

Calculations:

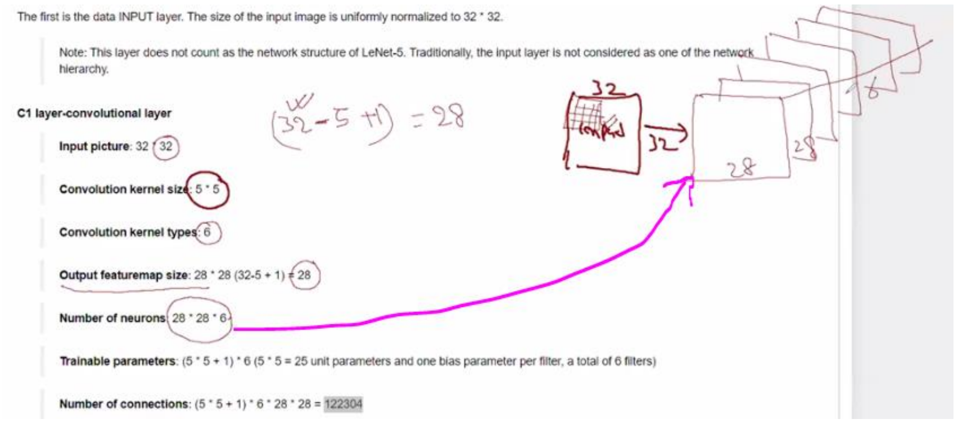


C1 layer-convolutional layer

- **Input picture**: 32 * 32
- **Convolution kernel size**: 5 * 5
- **Convolution kernel types**: 6
- **Output featuremap size**: 28 * 28 (32-5 + 1) = 28
- **Number of neurons**: 28 * 28 * 6
- **Trainable parameters**: (5 * 5 + 1) * 6 (5 * 5 = 25 unit parameters and one bias parameter per filter, a total of 6 filters)
- **Number of connections**: (5 * 5 + 1) * 6 * 28 * 28 = 122304

Here 32 is the size of the input

5 is the size of the kernel



The first is the data INPUT layer. The size of the input image is uniformly normalized to 32 * 32.

Note: This layer does not count as the network structure of LeNet-5. Traditionally, the input layer is not considered as one of the network hierarchy.

C1 layer-convolutional layer

Input picture: 32 * 32

Convolution kernel size: 5 * 5

Convolution kernel types: 6

Output featuremap size: 28 * 28 (32-5 + 1) = 28

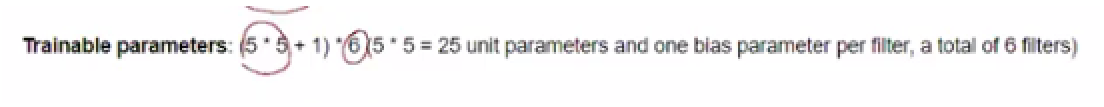
Number of neurons: 28 * 28 * 6

Trainable parameters: (5 * 5 + 1) * 6 (5 * 5 = 25 unit parameters and one bias parameter per filter, a total of 6 filters)

Number of connections: (5 * 5 + 1) * 6 * 28 * 28 = 122304

How many learnable parameters we have here.

The trainable parameter is nothing but the kernel size (5X5)



Trainable parameters: (5 * 5 + 1) * 6 (5 * 5 = 25 unit parameters and one bias parameter per filter, a total of 6 filters)

Here 6 is total feature.

Trainable parameter – Which is nothing but a things that I’m trying to learn or the pattern which I’m trying to learn.

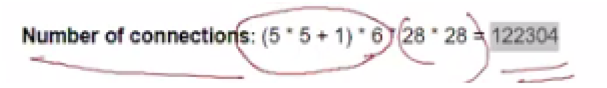
In case of a CNN the trainable parameter is the kernel. Because we don’t know what are the values we can consider in the kernel suppose we 5X5 we don’t know what are the values will be there in that kernel, So this are the trainable parameter with respect to CNN. This are the trainable parameter which is trained each and every iteration.

For that we trying to create a filters of size 5X5 of total 6 filters and in every kernel we are applying the biases called +1 in every layer wise.

(5X5+1)*6

Number of connections:

Number of permutation combinations between two layers.



Number of connections: (5 * 5 + 1) * 6 * 28 * 28 = 122304

S2 layer-pooling layer (downsampling layer)

Input: 28 * 28

Sampling area: 2 * 2

Sampling method: 4 inputs are added, multiplied by a trainable parameter, plus a trainable offset. Results via sigmoid

Sampling type: 6

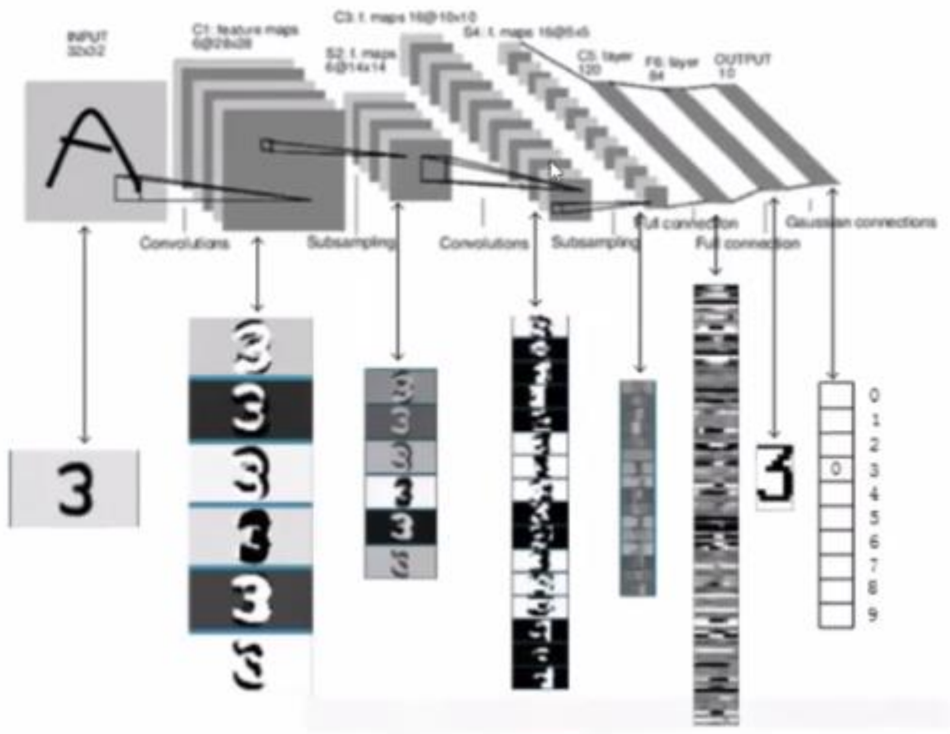
Output featureMap size: 14 * 14 (28/2)

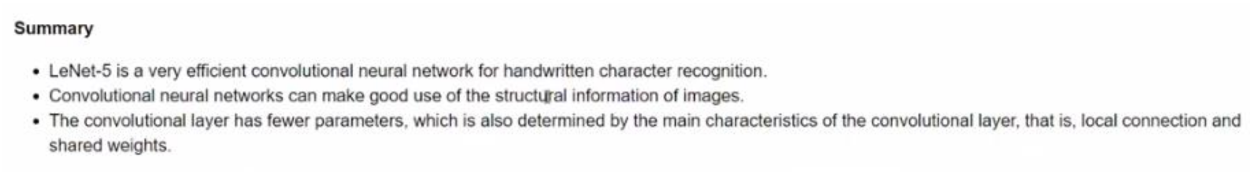
Number of neurons: 14 * 14 * 6

Trainable parameters: 2 * 6 (the weight of the sum + the offset)

Number of connections: (2 * 2 + 1) * 6 * 14 * 14

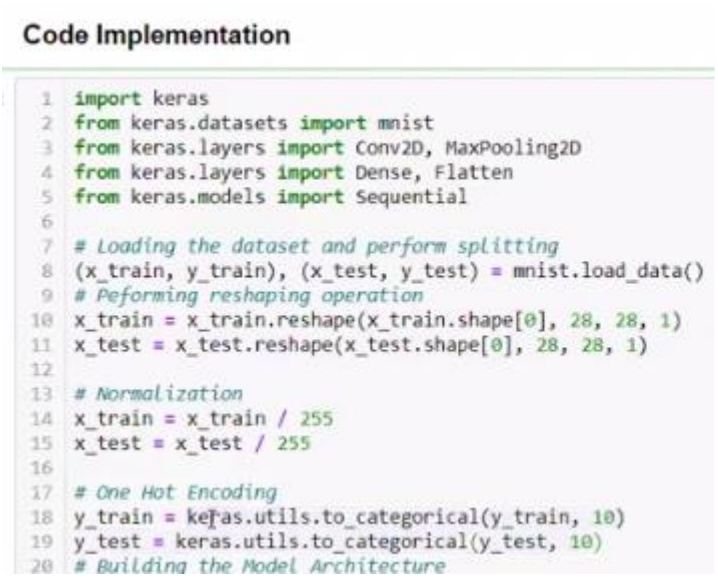
The size of each feature map in S2 is 1/4 of the size of the feature map in C1.




Summary

- LeNet-5 is a very efficient convolutional neural network for handwritten character recognition.
- Convolutional neural networks can make good use of the structural information of images.
- The convolutional layer has fewer parameters, which is also determined by the main characteristics of the convolutional layer, that is, local connection and shared weights.

.

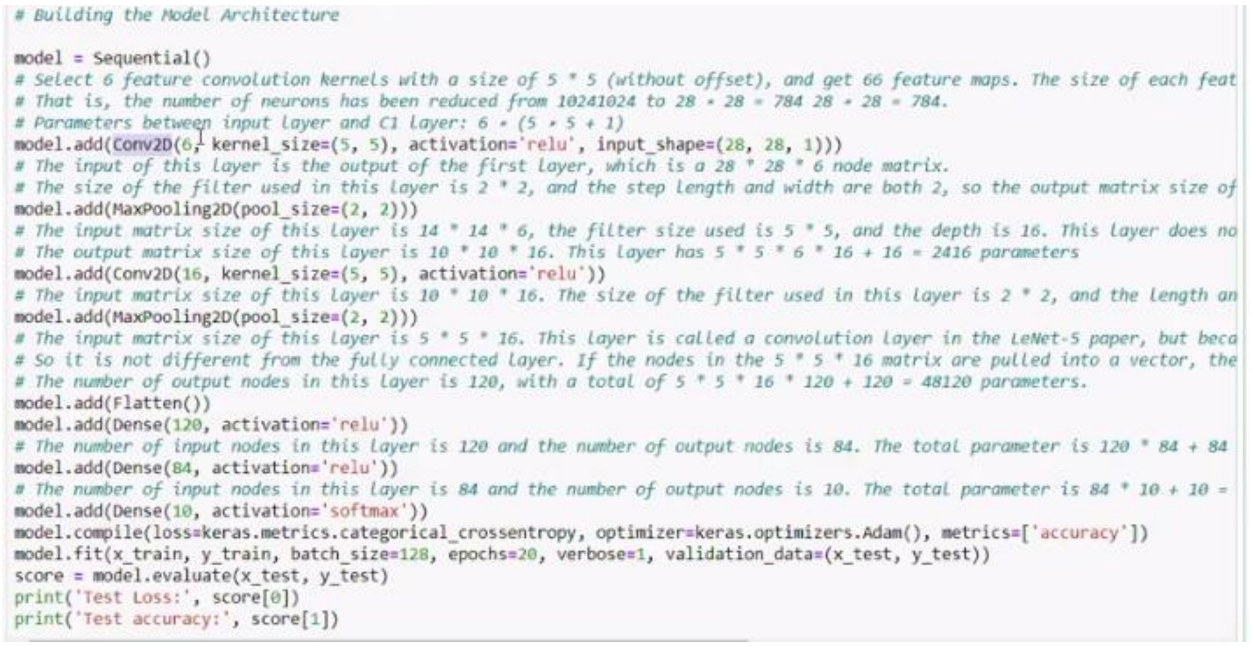

Code Implementation

```python
import keras
from keras.datasets import mnist
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Dense, Flatten
from keras.models import Sequential

# Loading the dataset and perform splitting
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Peforming reshaping operation
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

# Normalization
x_train = x_train / 255
x_test = x_test / 255

# One Hot Encoding
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
# Building the Model Architecture
```

Here I'm going to take the mnist dataset which is already available in keras library.

After that I'm going to reshape the dataset.

Normalization – Convert colorful dataset into grayscale.



```python
# Building the Model Architecture

model = Sequential()
# Select 6 feature convolution kernels with a size of 5 * 5 (without offset), and get 66 feature maps. The size of each feat
# That is, the number of neurons has been reduced from 10241024 to 28 * 28 = 784 28 * 28 = 784.
# Parameters between input layer and C1 layer: 6 * (5 * 5 + 1)
model.add(Conv2D(6, kernel_size=(5, 5), activation='relu', input_shape=(28, 28, 1)))
# The input of this layer is the output of the first layer, which is a 28 * 28 * 6 node matrix.
# The size of the filter used in this layer is 2 * 2, and the step length and width are both 2, so the output matrix size of
model.add(MaxPooling2D(pool_size=(2, 2)))
# The input matrix size of this layer is 14 * 14 * 6, the filter size used is 5 * 5, and the depth is 16. This layer does no
# The output matrix size of this layer is 10 * 10 * 16. This layer has 5 * 5 * 6 * 16 + 16 = 2416 parameters
model.add(Conv2D(16, kernel_size=(5, 5), activation='relu'))
# The input matrix size of this layer is 10 * 10 * 16. The size of the filter used in this layer is 2 * 2, and the length an
model.add(MaxPooling2D(pool_size=(2, 2)))
# The input matrix size of this layer is 5 * 5 * 16. This layer is called a convolution layer in the LeNet-5 paper, but beca
# So it is not different from the fully connected layer. If the nodes in the 5 * 5 * 16 matrix are pulled into a vector, the
# The number of output nodes in this layer is 120, with a total of 5 * 5 * 16 * 120 + 120 = 48120 parameters.
model.add(Flatten())
model.add(Dense(120, activation='relu'))
# The number of input nodes in this layer is 120 and the number of output nodes is 84. The total parameter is 120 * 84 + 84
model.add(Dense(84, activation='relu'))
# The number of input nodes in this layer is 84 and the number of output nodes is 10. The total parameter is 84 * 10 + 10 =
model.add(Dense(10, activation='softmax'))
model.compile(loss=keras.metrics.categorical_crossentropy, optimizer=keras.optimizers.Adam(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=128, epochs=20, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test)
print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

Here we have used relu as a activation function because that time relu was not available. So that why they have used tanh function.