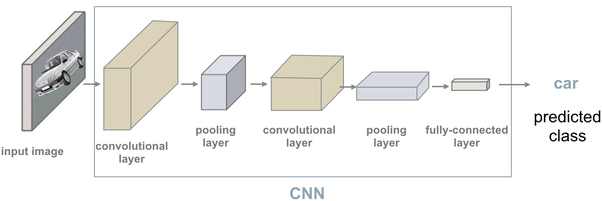
1)explain the convolutional neural networks & how does it work ?

Ans : Convolutional neural networks (CNNs) are specialized neural networks that are specifically designed to capture localized (spatial) information in a dataset. By explicitly encoding that in the architecture, CNNs are designed to better handle image (2D) data, and by extension, 1D or 3D data.



convolutional neural network architecture generally has several components:

A convolution layer - you can think of this layer as “what relevant features are we picking up in an image?” In a convolutional neural network, we have multiple convolutional layers that extract low to high-level features depending on what specific layer we are focusing on. To give an (over-simplified) intuition, earlier convolutional layers pick up lower-level features (i.e. like lines and edges) while later convolutional layers pick up higher-level features based on inputs from lower-level features (i.e. shapes, structures) - analogous to how vision works in the human brain.

A pooling layer - convolutional neural networks are typically used for image classification. However, images are high-dimensional data so we would prefer to reduce the dimensionality to minimize the possibility of overfitting. Pooling essentially reduces the spatial dimensions of the image based on certain mathematical operations such as average or max-pooling (there’s a nice graphic here). We generally incorporate pooling since it (1) generally acts as a noise suppressant (2) makes it invariant to translation movement for image classification and (3) helps capture essential structural features of the represented images without being bogged down by the fine details.

Fully-connected layer - You can think of a series of convolution and pooling operations as dimensionality reduction steps prior to passing this information over to the fully connected (Dense) layer. Essentially, what the fully connected layer does is that it takes the "compressed" representation of the image and it tries to fit a basic NN (multi-layer perceptron) when doing classification.

2)how does refactoring parts of your neural networks definition favor you ?

Ans : Deep neural networks (DNN) are growing in capability and applicability. Their effectiveness has led to their use in safety critical and autonomous systems, there is a cost-effective methods available for reasoning about the behavior of a DNN. A DNN refactoring defines the transformation of the DNN's architecture, the number and size of its layers, and the distillation of the learned relationships between the input features and function outputs of the original to train the transformed network. Unlike with traditional code refactoring, DNN refactoring does not guarantee functional equivalence of the two networks, but rather it aims to preserve the accuracy of the original network while producing a simpler network that is amenable to more efficient property verification. We present an automated framework for DNN refactoring.

3)what does it mean to filters ?is it necessary to include it in the MNIST CNN ?what is the reason for this ?

Ans :filter acts as a single template or pattern, which, when convolved across the input, finds similarities between the stored template & different locations/regions in the input image.

MNIST is a Modified National Institute of Standards and Technology.MNIST is a large database of small, square 28x28 pixel grayscale images of handwritten single digits between 0 and 9. It consists of a total of 70,000 handwritten images of digits, with the training set having 60,000 images and the test set having 10,000. All images are labeled with the respective digit that they represent. There are a total of 10 classes of digits (from 0 to 9)

mnist dataset is a dataset of handwritten images . We can get 99.06% accuracy by using CNN Convolutional Neural Network with a functional model. The reason for using a functional model is to maintain easiness while connecting the layers so it is necessary to include it in MNIST CNN.

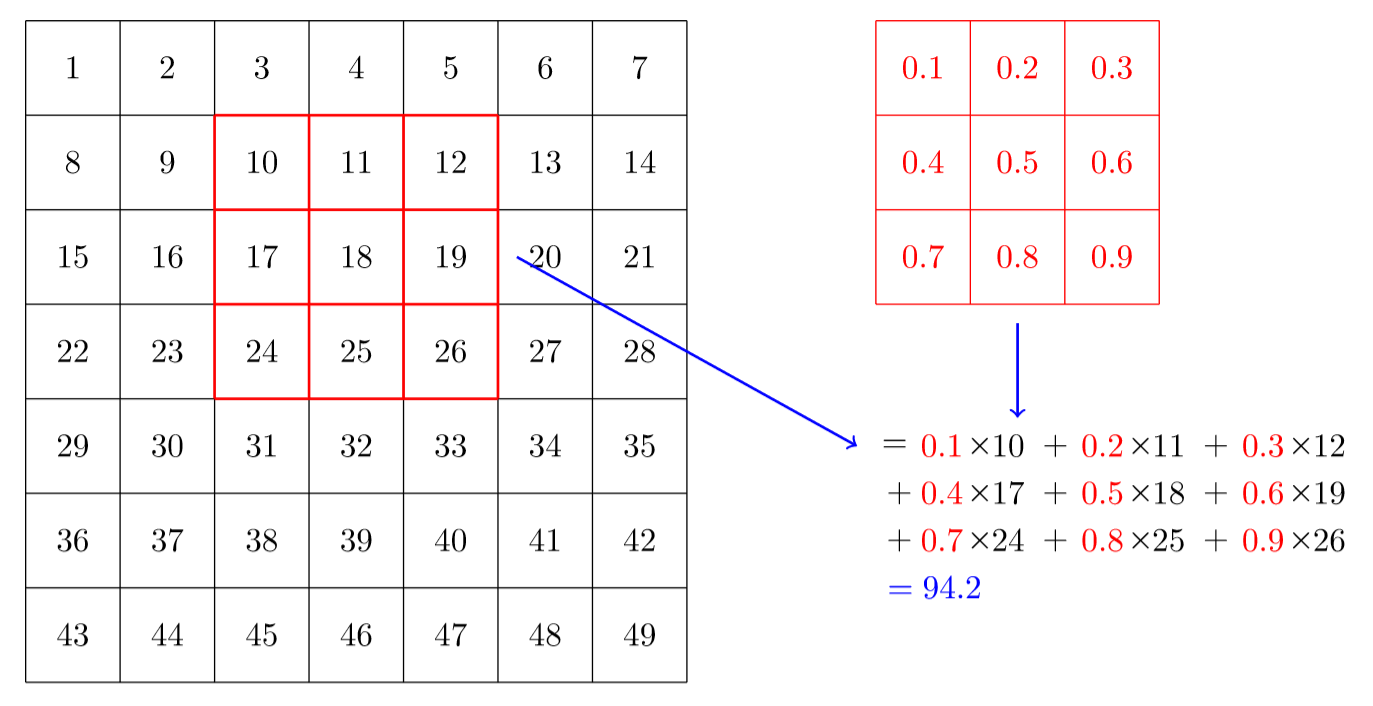
4)what exactly does NCHW stand for ?

Ans : NCHW stand for a data whose layout is batch\_size, channel, height, width.

C-channel,H-hight,W-weight.N-batch.

5)why are there,7\*7(1168-16)Multiplication in the MNIST CNN'S third layer ?

ans :



convolution applies a kernel across an image. A kernel is a little matrix, such as the 3×3 matrix in the top right of <>.</p> </div> </div> </div>

The 7×7 grid to the left is the image we're going to apply the kernel to. The convolution operation multiplies each element of the kernel by each element of a 3×3 block of the image. The results of these multiplications are then added together. The diagram in <> shows an example of applying a kernel to a single location in the image, the 3×3 block around cell 18.</p>

Let's do this with code. First, we create a little 3×3 matrix like so:

</div> </div> </div>

top\_edge = tensor([[-1,-1,-1],

[ 0, 0, 0],

[ 1, 1, 1]]).float()

We're going to call this our kernel (because that's what fancy computer vision researchers call these). And we'll need an image.

path = untar\_data(URLs.MNIST\_SAMPLE)

im3 = Image.open(path/'train'/'3'/'12.png')

show\_image(im3);

Now we're going to take the top 3×3-pixel square of our image, and multiply each of those values by each item in our kernel. Then we'll add them up, like so:

im3\_t = tensor(im3)

im3\_t[0:3,0:3] \* top\_edge

tensor([[-0., -0., -0.],

[0., 0., 0.],

[0., 0., 0.]])

(im3\_t[0:3,0:3] \* top\_edge).sum()

tensor(0.)

6)explain the definition of receptive field.

Ans :The receptive field is perhaps one of the most important concepts in Convolutional Neural Networks (CNNs) that deserves more attention from the literature. All of the state-of-the-art object recognition methods design their model architectures around this idea.The receptive field is defined as the region in the input space that a particular CNN’s feature is looking at, a receptive field of a feature can be described by its center location and its size.Within a receptive field, the closer a pixel to the center of the field, the more it contributes to the calculation of the output feature. Which means that a feature does not only look at a particular region.

7)what is the scale of an activations receptive field after two stride-2 convolutions ?what is the reason for this ?

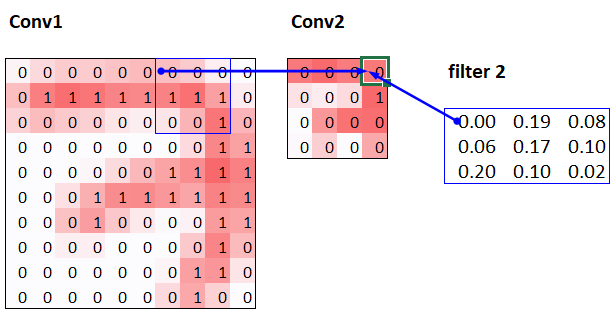
Ans :When we use a stride-2 convolution, we often increase the number of features because we're decreasing the number of activations in the activation map by a factor of 4; we don't want to decrease the capacity of a layer by too much at a time."

There is one bias for each channel. (Sometimes channels are called features or filters when they are not input channels.) The output shape is 64x4x14x14, and this will therefore become the input shape to the next layer. The next layer, according to summary, has 296 parameters. Let's ignore the batch axis to keep things simple. So for each of 14\*14=196 locations we are multiplying 296-8=288 weights (ignoring the bias for simplicity), so that's 196\*288=56\_448 multiplications at this layer. The next layer will have 7\*7\*(1168-16)=56\_448 multiplications.

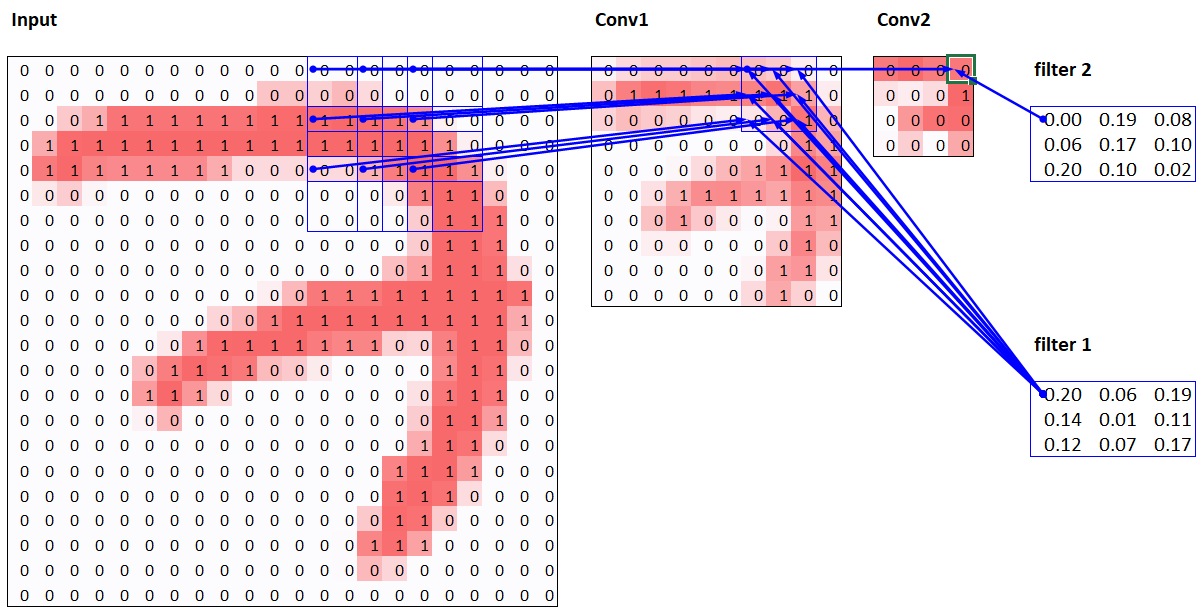
What happened here is that our stride-2 convolution halved the grid size from 14x14 to 7x7, and we doubled the number of filters from 8 to 16, resulting in no overall change in the amount of computation. If we left the number of channels the same in each stride-2 layer, the amount of computation being done in the net would get less and less as it gets deeper. But we know that the deeper layers have to compute semantically rich features (such as eyes or fur), so we wouldn't expect that doing less computation would make sense.

Another way to think of this is based on receptive fields.

The receptive field is the area of an image that is involved in the calculation of a layer. you'll find an Excel spreadsheet called conv-example.xlsx that shows the calculation of two stride-2 convolutional layers using an MNIST digit. Each layer has a single kernel. <> shows what we see if we click on one of the cells in the conv2 section, which shows the output of the second convolutional layer, and click trace precedents.</p> </div> </div> </div>



Here, the cell with the green border is the cell we clicked on, and the blue highlighted cells are its precedents—that is, the cells used to calculate its value. These cells are the corresponding 3×3 area of cells from the input layer (on the left), and the cells from the filter (on the right). Let's now click trace precedents again, to see what cells are used to calculate these inputs. <> shows what happens.</p> </div> </div> </div>



this example, we have just two convolutional layers, each of stride 2, so this is now tracing right back to the input image. We can see that a 7×7 area of cells in the input layer is used to calculate the single green cell in the Conv2 layer. This 7×7 area is the receptive field in the input of the green activation in Conv2. We can also see that a second filter kernel is needed now, since we have two layers.

As you see from this example, the deeper we are in the network (specifically, the more stride-2 convs we have before a layer), the larger the receptive field for an activation in that layer. A large receptive field means that a large amount of the input image is used to calculate each activation in that layer is. We now know that in the deeper layers of the network we have semantically rich features, corresponding to larger receptive fields. Therefore, we'd expect that we'd need more weights for each of our features to handle this increasing complexity. This is another way of saying the same thing we mentioned in the previous section: when we introduce a stride-2 conv in our network, we should also increase the number of channels.

When writing this particular chapter, we had a lot of questions we needed answers for, to be able to explain CNNs to you as best we could. Believe it or not, we found most of the answers on Twitter. We're going to take a quick break to talk to you about that now, before we move on to color images.

.

8)what is the tensor Representation of color image ?

Ans : Tensors are collections of data in a structured type that are optimized as numbers to be ready for calculations. They’re similar to working with multidimensional arrays in JavaScript.

they’re created to do many side by side calculations at once and can batch process at high speeds. They also provide us with direct access to data in a useable format. Outside of Machine Learning, there are libraries that use tensors for images, sound, 3d models, etc.we know that

those small units create an image. Each of these pixels holds a position in the image, and each pixel can be represented by an array of three numbers that correspond with an RGB (red, green, blue) value that represents a color.When we transform an image into a tensor, each of those pixels in the position they hold in the image, get transformed into tensors. Once we transform these tensors, we can train our models with this data, among other things. We can mirror images, resize, crop, and manipulate in other ways that are useful to us.

9)how does a color input Interact with a convolution ?

Ans :In a convolutional layer, each neuron receives input from only a restricted area of the previous layer called the neuron's receptive field. Typically the area is a square (e.g. 5 by 5 neurons). Whereas, in a fully connected layer, the receptive field is the entire previous layer.Naturally, CNN is designed to learn classification method based on shape information, but we proved that CNN can also learn classification based on color distribution. In our method, we convert the input image to two different color spaces, HSV and CIE Lab, and run it to some CNN architecture.

When RGB image is used as input to CNN, the depth of filter (or kernel) is always equal to depth of image (so in case of RGB, that is 3). So, If 32x32x3 is the input image, the filter has to be NxNx3 (where N is height and width of filter like 3x3x3). Therefore, the filter has 3 two dimensional matrices.