What is going on in the model with two convolution layers?

```
model = torch.nn.Sequential(
```

We begin by setting up the sequential API. Remember our input data have shape $1 \times 28 \times 28$ (b/w, width and height are both 28)

```
    torch.nn.Conv2d(1, 20, 5, 1, 0),
```

The first convolution layer takes in a $1 \times 28 \times 28$ image, and produce 20 *feature maps*. The kernel size is 5 and stride is 1, so each feature map will have width and height $28 - 5 + 1 = 24$

```
    torch.nn.ReLU(),
```

The ReLU activation doesn't change dimensionality.

```
    torch.nn.MaxPool2d(2),
```

Pooling with 2×2 filters reduce our maps to shape $1 \times 12 \times 12$, we still have 20 of these maps to pass on.

```
    torch.nn.Conv2d(20, 40, 5, 1, 0),
```

Our next convolution layer takes all 20 feature maps and input. With a stride of 5, this reduces the size to $12 - 5 + 1 = 8$. We make 40 new maps in this layer

```
    torch.nn.ReLU(),
```

ReLU once again doesn't change anything

```
    torch.nn.MaxPool2d(2),
```

Pooling reduces each of the 40 maps to size 4×4

```
    Flatten(),
```

Now we want to convert the tensor into a vector which can be passed to a vanilla neural layer. The output will have one element per pixel (4×4) per map (40)

```
    torch.nn.Linear(4 * 4 * 40, 1000),
```

Now we pass the $4 \times 4 \times 40$ element vector to a linear layer with 1000 output features

```
    torch.nn.ReLU(),
```

ReLU once again

```
    torch.nn.Dropout(0.5),
```

Weight dropout to avoid overfitting

```
    torch.nn.Linear(1000, 1000),
```

Another linear layer with as many outputs as we provide input nodes

```
    torch.nn.Dropout(0.5),
```

Another layer of dropouts

```
    torch.nn.Linear(1000, 10),
```

Now we take the 1000 outputs from the last layer and collapse down to 10 nodes, one for each digit

```
    torch.nn.Softmax()
)
```

The softmax function finally identifies the most likely digit.