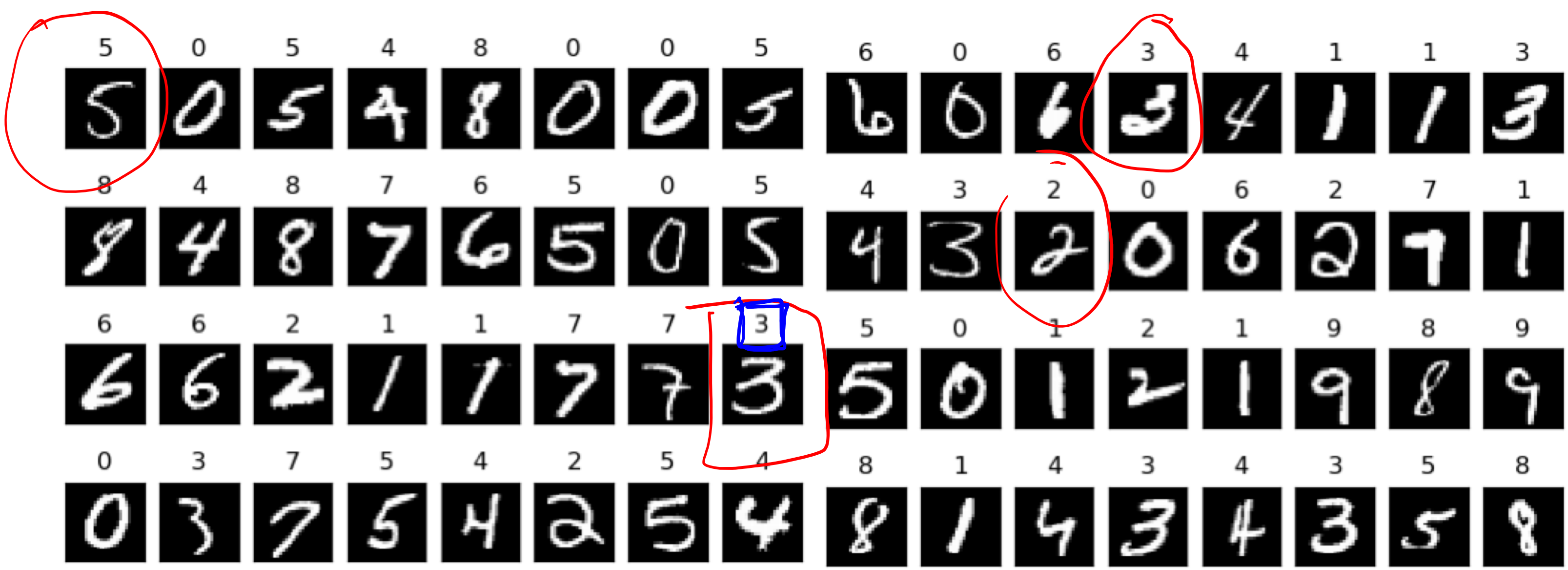
Identifying Digits with a CNN

Heni Ben Amor, Ph.D.
Assistant Professor
Arizona State University

Ira A. Fulton Schools of Engineering
Arizona State University

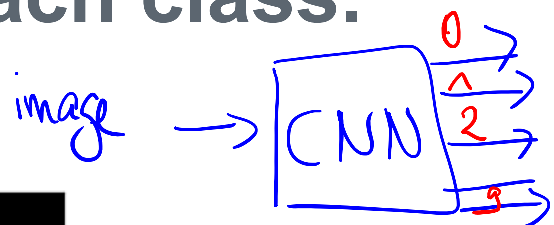
MNIST Database

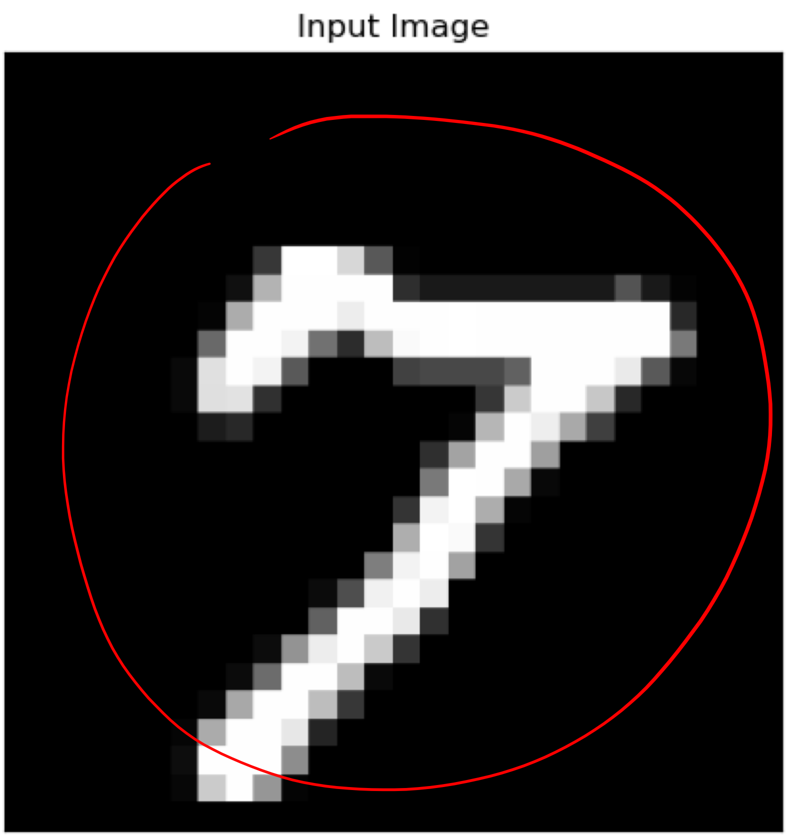
- Handwritten labeled digits
- 60,000 sample training set
- 10,000 sample testing set
- Available online or through PyTorch's torchvision package



Our Model

- Take in an input image and output a log probability for each class.





0: -42.6972
1: -46.8689
2: -29.2313
3: -21.1472
4: -27.0222
5: -23.9541
6: -60.7476
7: 0.0000
8: -25.9002
9: -17.0474

Log Probabilities

- It is often useful to take the log of our probabilities.
- More efficient computation, addition replaces multiplication operation.

$$\log(x \cdot y) = \underline{\log(x)} + \underline{\log(y)}$$

- For the following examples probabilities are represented as:

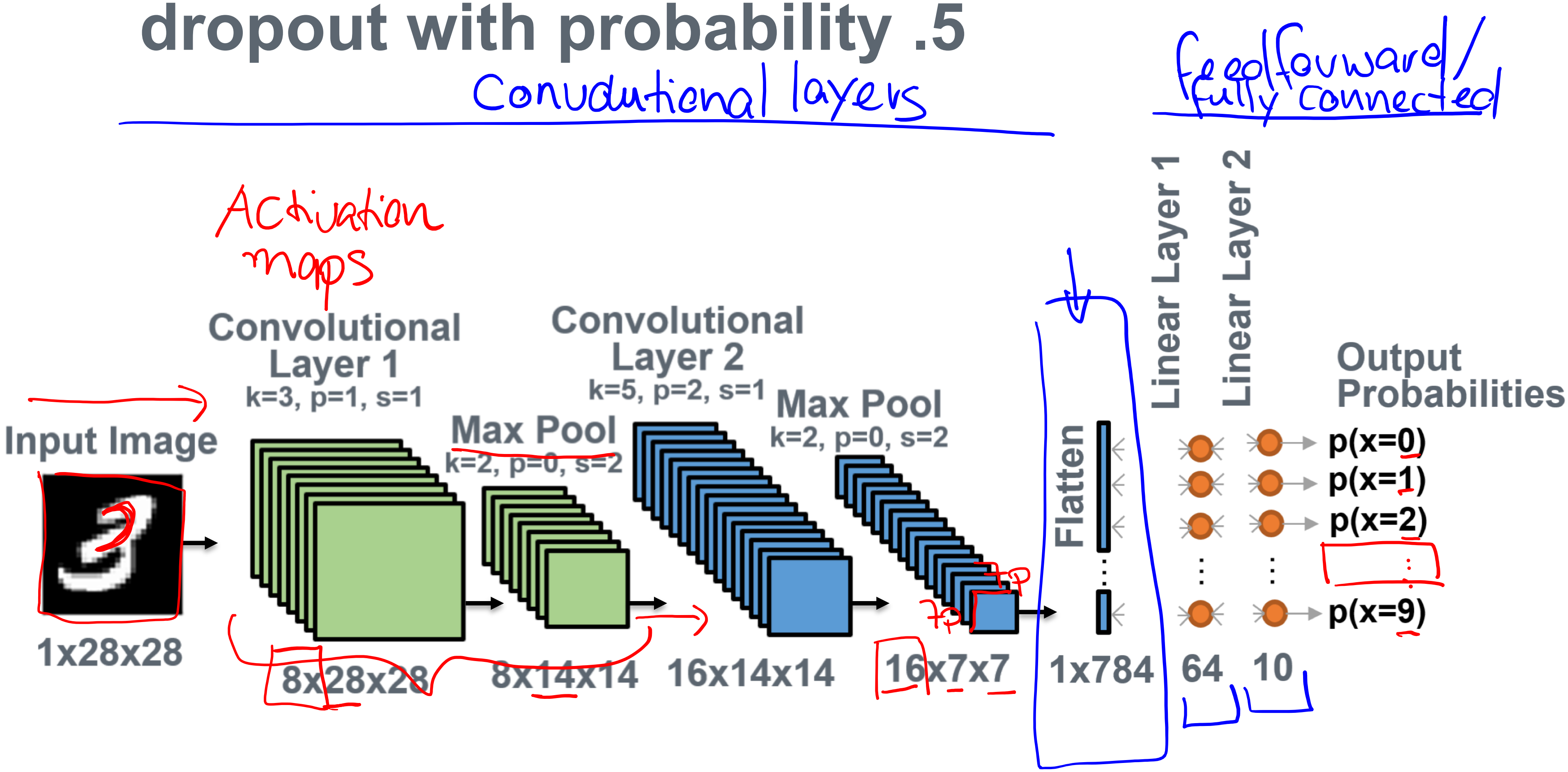
log-probability

$$\boxed{p'} = log_{\boxed{2}}(p)$$

- Yields only <u>negative values</u>
- Values closer to 0 are more likely

Example Network Architecture

- Two convolutional layer network using kernel size k, padding p, stride s, and dropout with probability .5



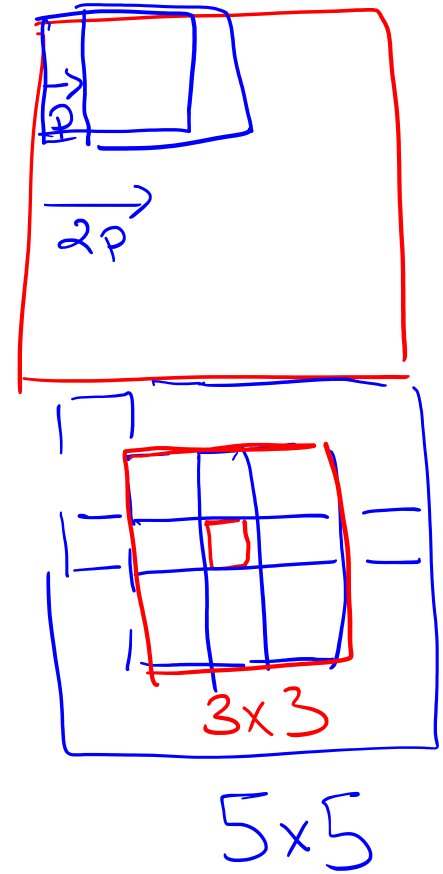
PyTorch Code

Conv

```
in_channels, cnn1_channels, cnn2_channels = 1, 8, 16
fc_hidden, num_classes = 64, 10

class CNN_MNIST(nn.Module):
  def __init__(self):
    super(CNN_MNIST, self).__init__()
    self.conv1 = nn.Conv2d(in_channels, cnn1_channels,
                           kernel_size=3, padding=1, stride=1)
    self.conv2 = nn.Conv2d(cnn1_channels, cnn2_channels,
                           kernel_size=5, padding=2, stride=1)
    self.dropout = nn.Dropout2d(p=.5)
    self.fc1 = nn.Linear(cnn2_channels*7*7, fc_hidden)
    self.fc2 = nn.Linear(fc_hidden, num_classes)
```

fully - connected



PyTorch Code

Conv 1 + Relu + MaxP

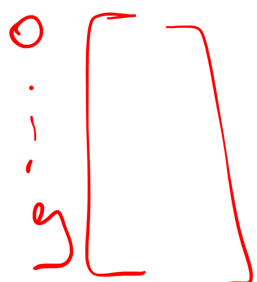
```
def forward(self, x):
    x = F.relu(self.conv1(x)) #8*28*28
    x = F.max_pool2d(x, kernel_size=2, stride=2, padding=0) #8*14*14
    x = self.dropout(x)
```

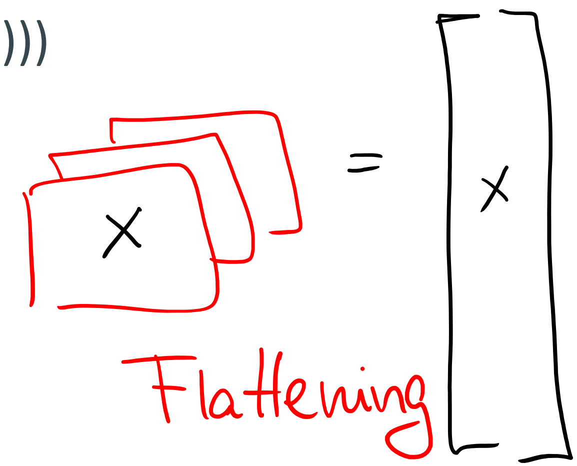
Conv2 + ReLU + Max

```
    x = F.relu(self.conv2(x)) #16*14*14
    x = F.max_pool2d(x, kernel_size=2, stride=2, padding=0) #16*7*7
    x = self.dropout(x)
```

Fully Connected

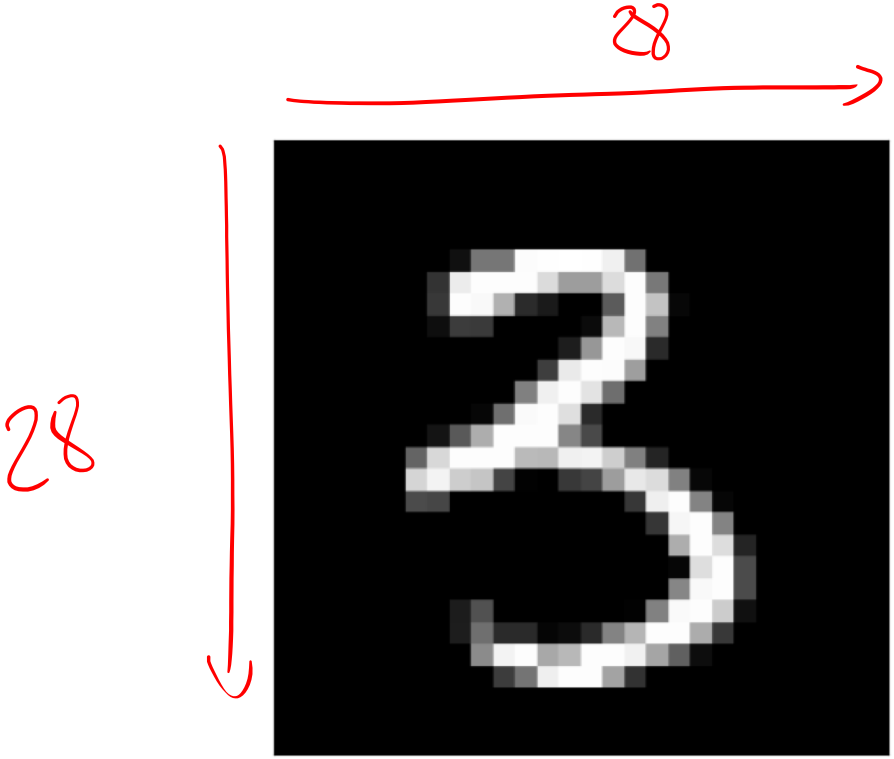
```
    x = F.relu(self.fc1(x.view(-1, cnn2_channels*7*7)))
    y = F.log_softmax(self.fc2(x), dim=1)
    return y
```





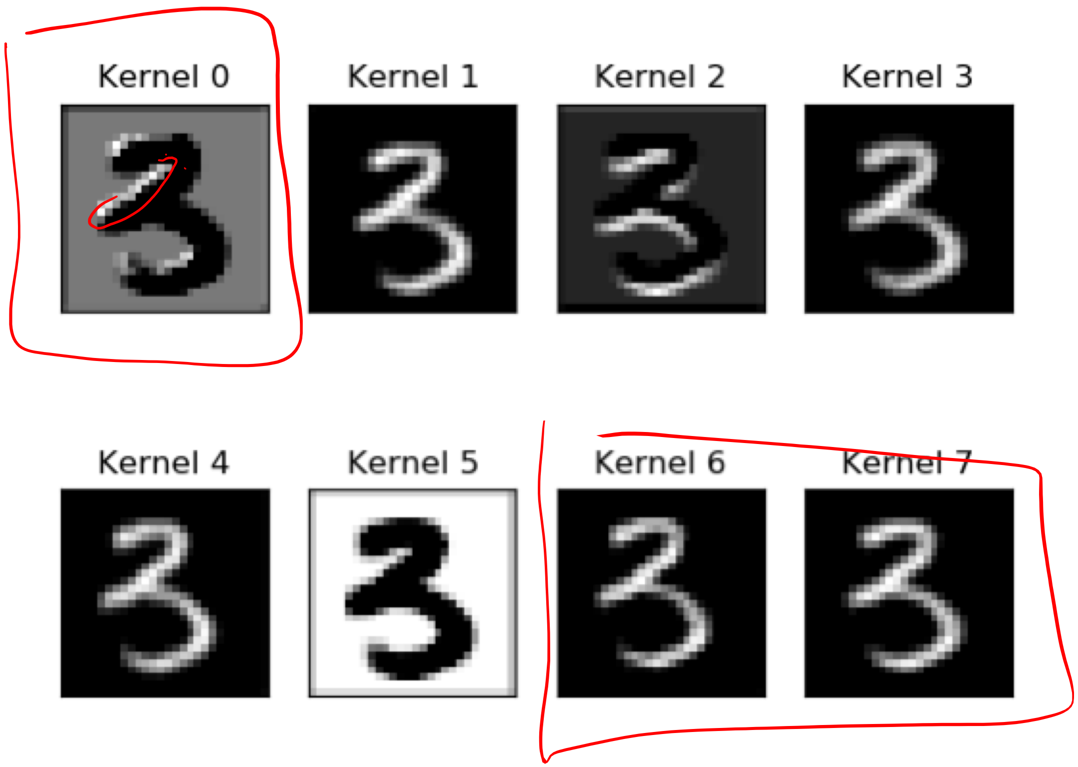
Flattening

Input Image



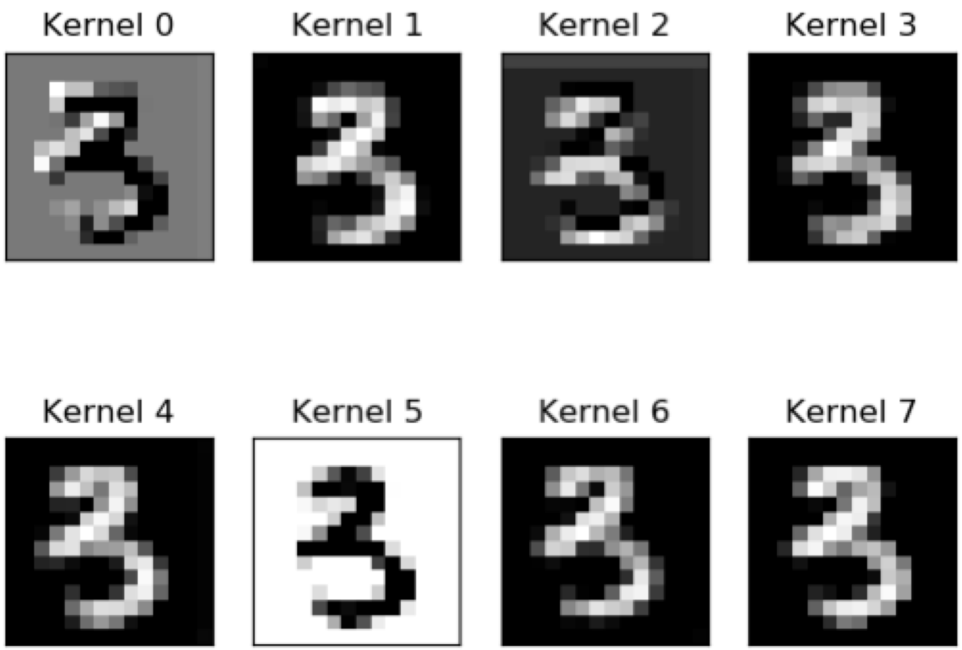
1x28x28

Convolutional Layer 1



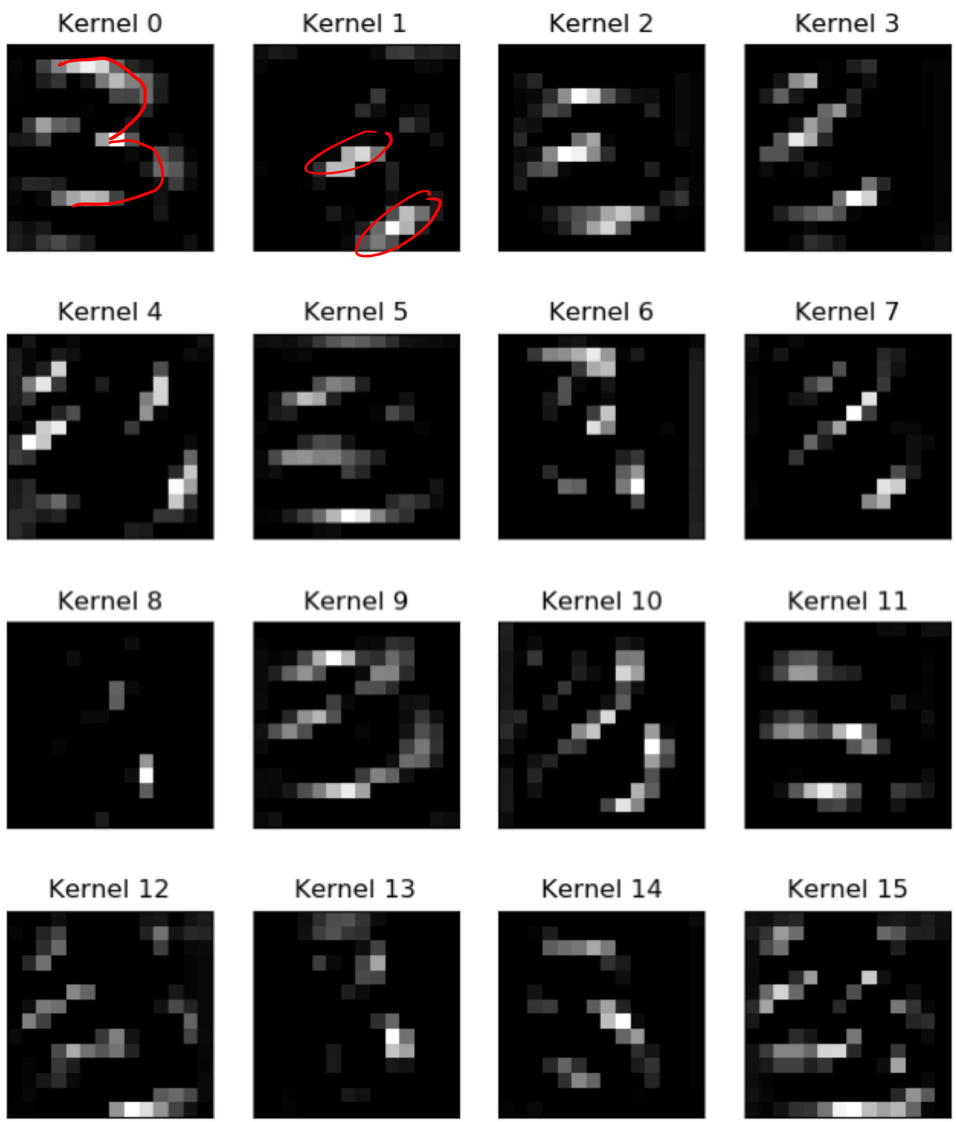
8x28x28

Max Pooled



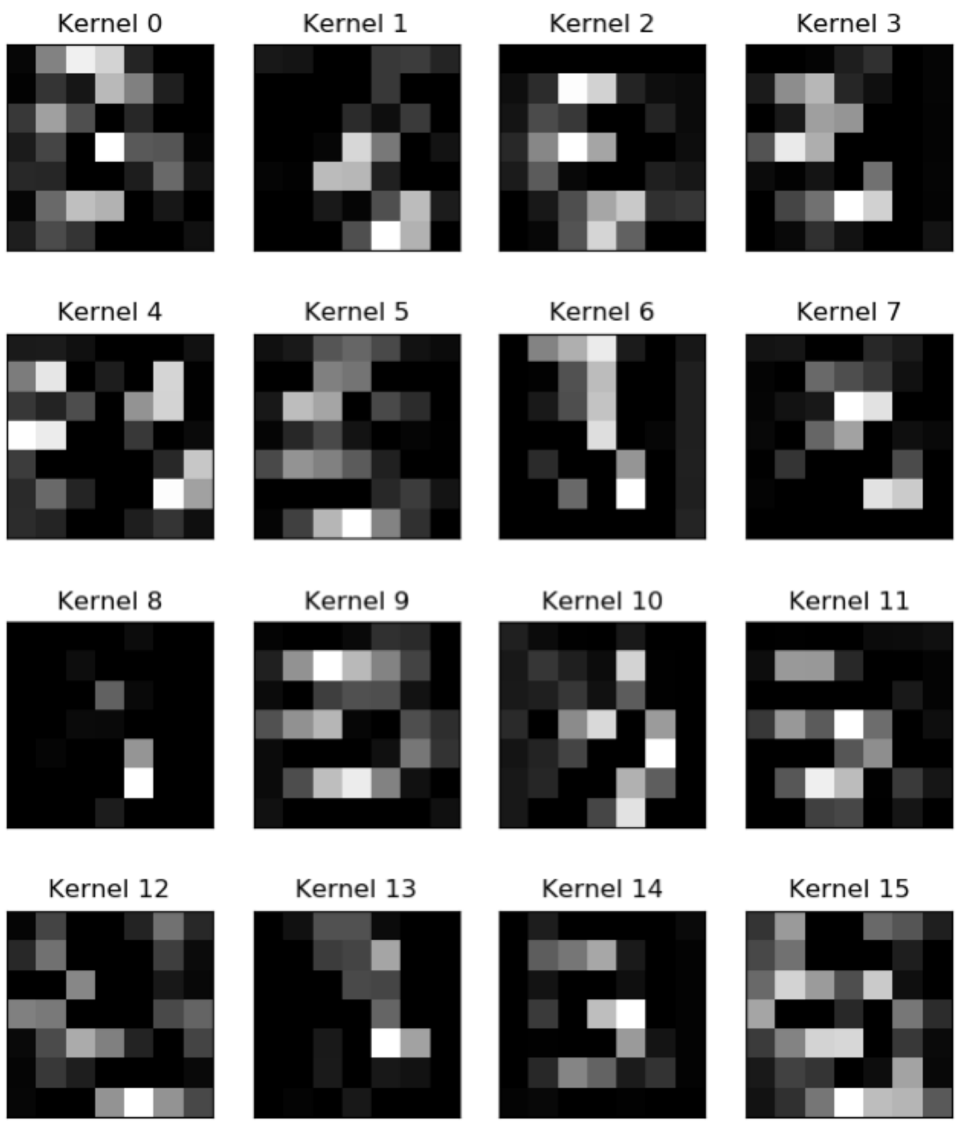
8x14x14

Convolutional Layer 2



16x14x14

Max Pooled



$x =$

16x7x7

Output

- Convolutional layer is flattened and passed through fully-connected layers to final output.

0: -63.8713	5: -24.0022
1: -39.7138	6: -57.5555
2: -34.8303	7: -35.8093
3: 0.0000	8: -22.0385
4: -47.8917	9: -32.1421

Summary

- MNIST dataset
- Log probability
- Example architecture
- Dimensionality changes throughout
- Convolutional layers visualized