With PyTorch 0.2.0_2

Lab 11

CNN MNIST

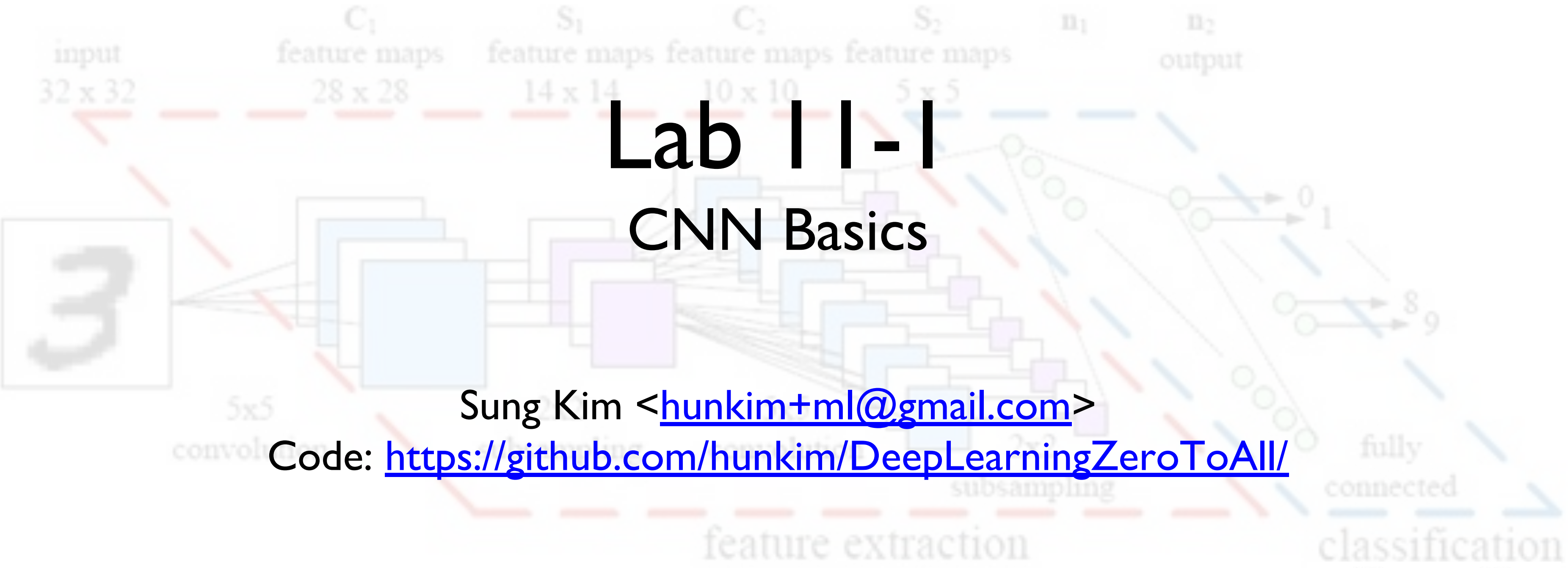
Sung Kim <hunkim+ml@gmail.com>

Code: https://github.com/hunkim/DeepLearningZeroToAll/

With PyTorch 0.2.0_2

Lab 11-1

CNN Basics



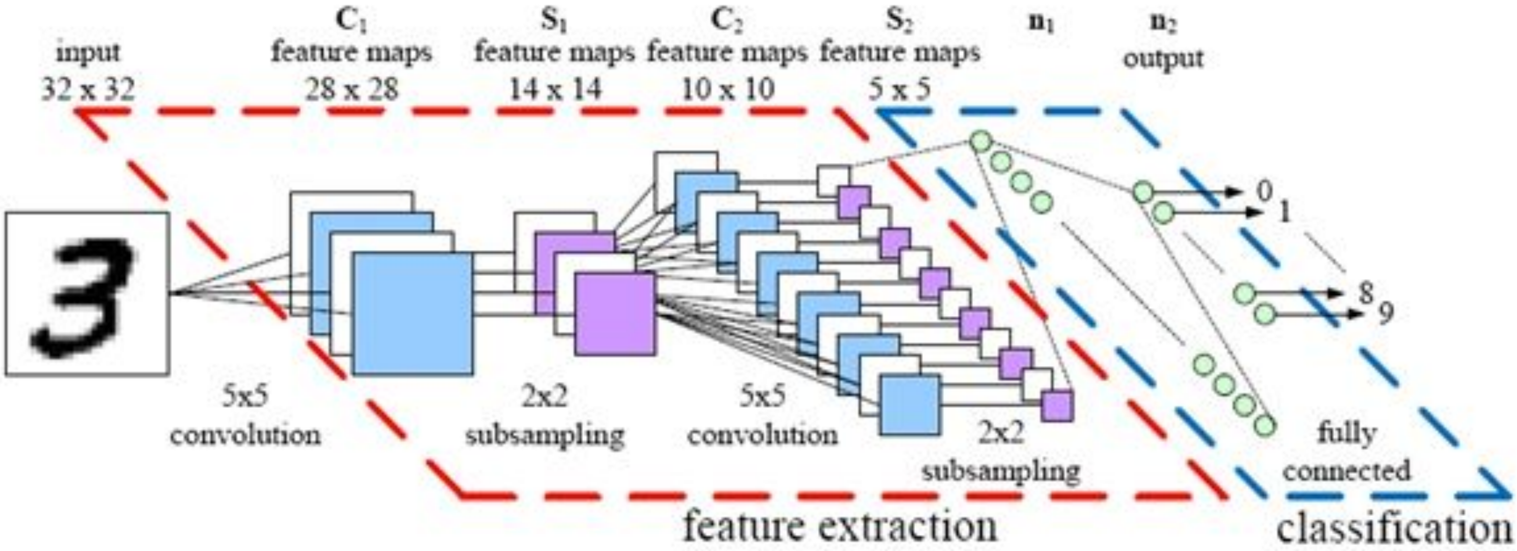
Sung Kim <hunkim+ml@gmail.com>

Code: https://github.com/hunkim/DeepLearningZeroToAll/

https://github.com/hunkim/DeepLearningZeroToAll/tree/master/pytorch

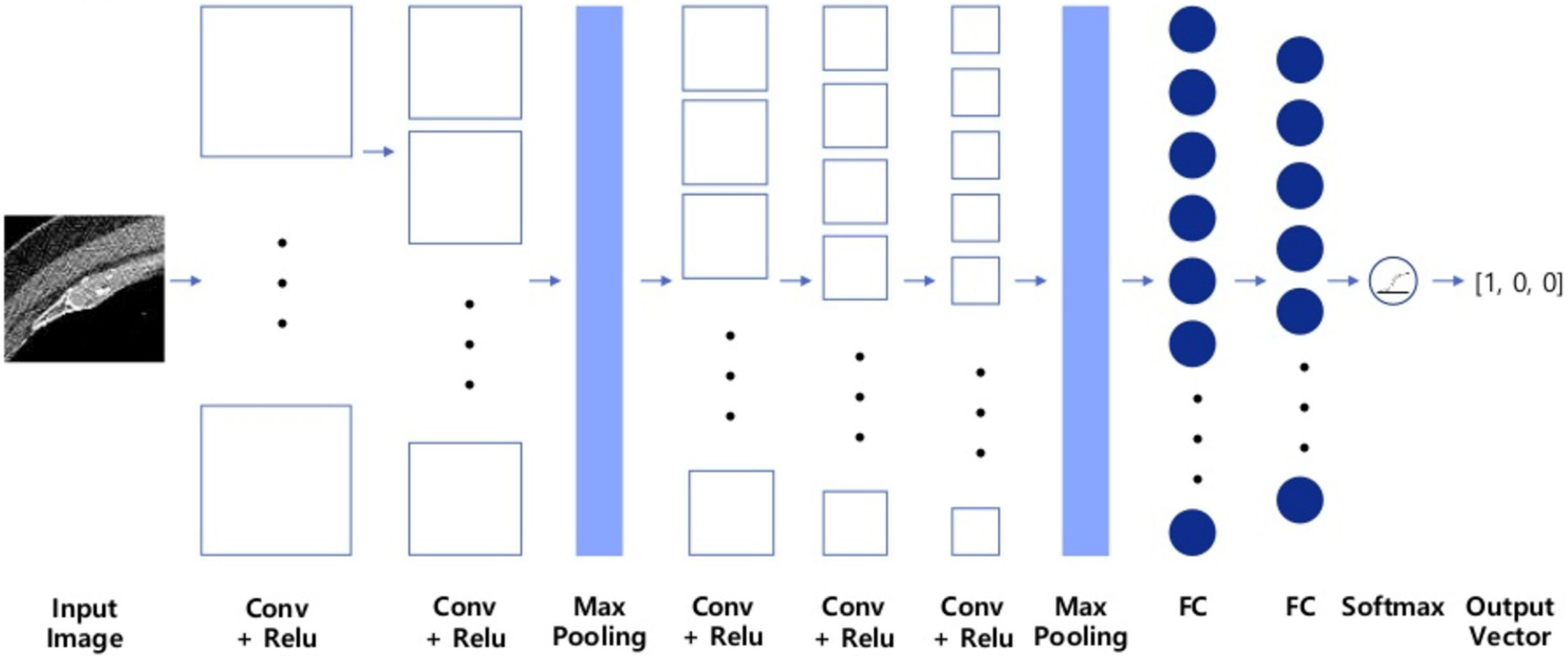
https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-11-1-mnist_cnn.py

CNN



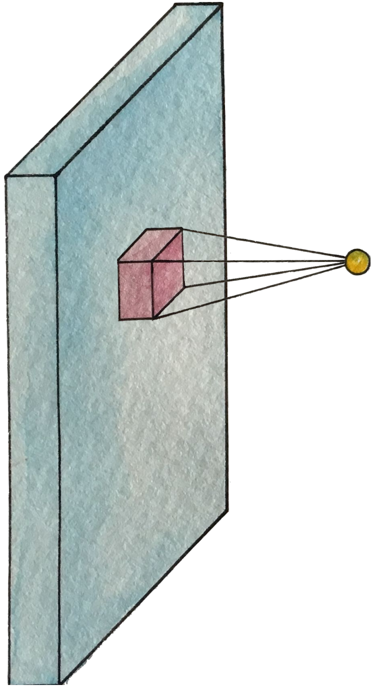
http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg

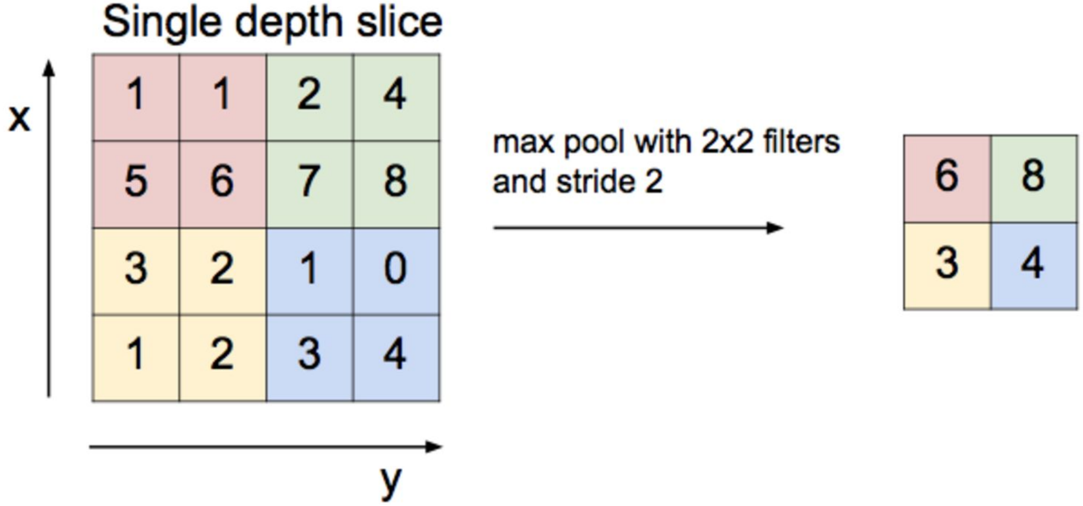
CNN for CT images



Asan Medical Center & Microsoft Medical Bigdata Contest Winner by GeunYoung Lee and Alex Kim
https://www.slideshare.net/GYLee3/ss-72966495

Convolution layer and max pooling





Single depth slice

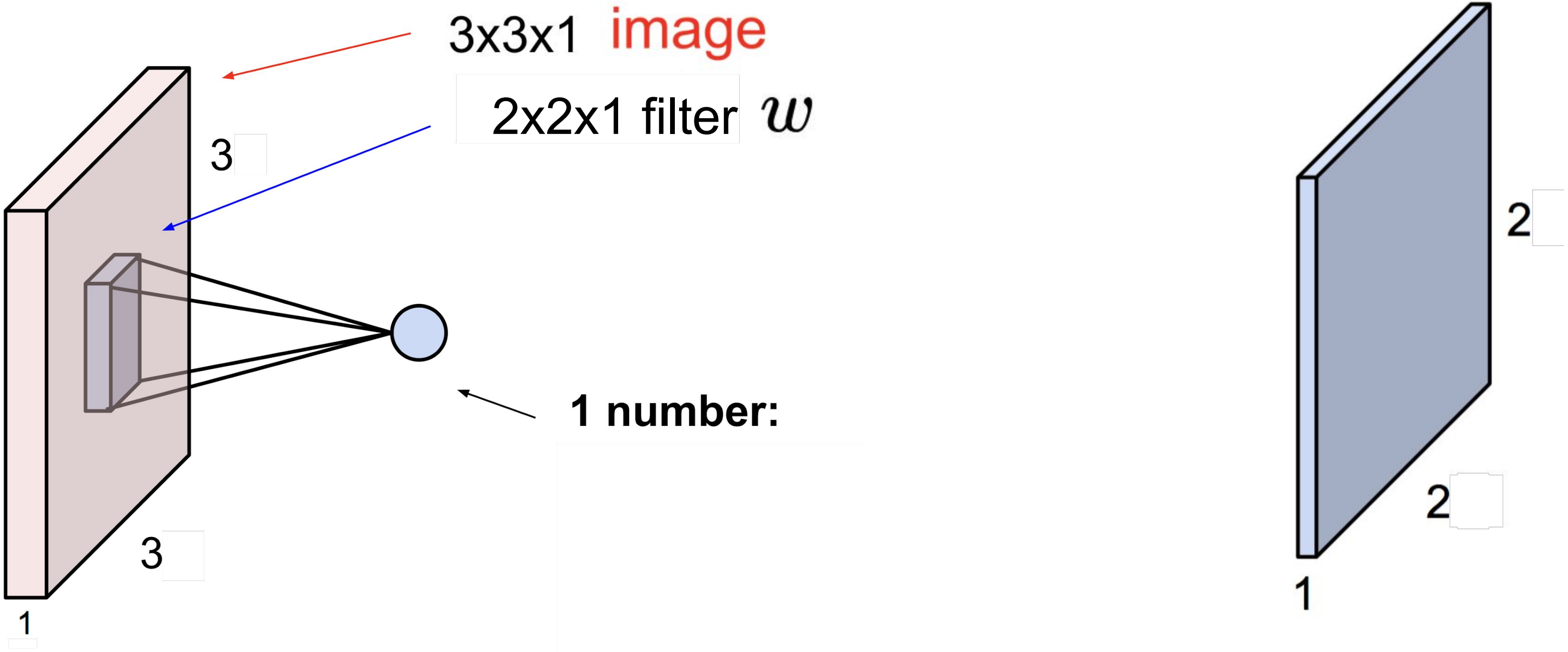
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters and stride 2

6	8
3	4

Simple convolution layer

`Stride: 1x1`

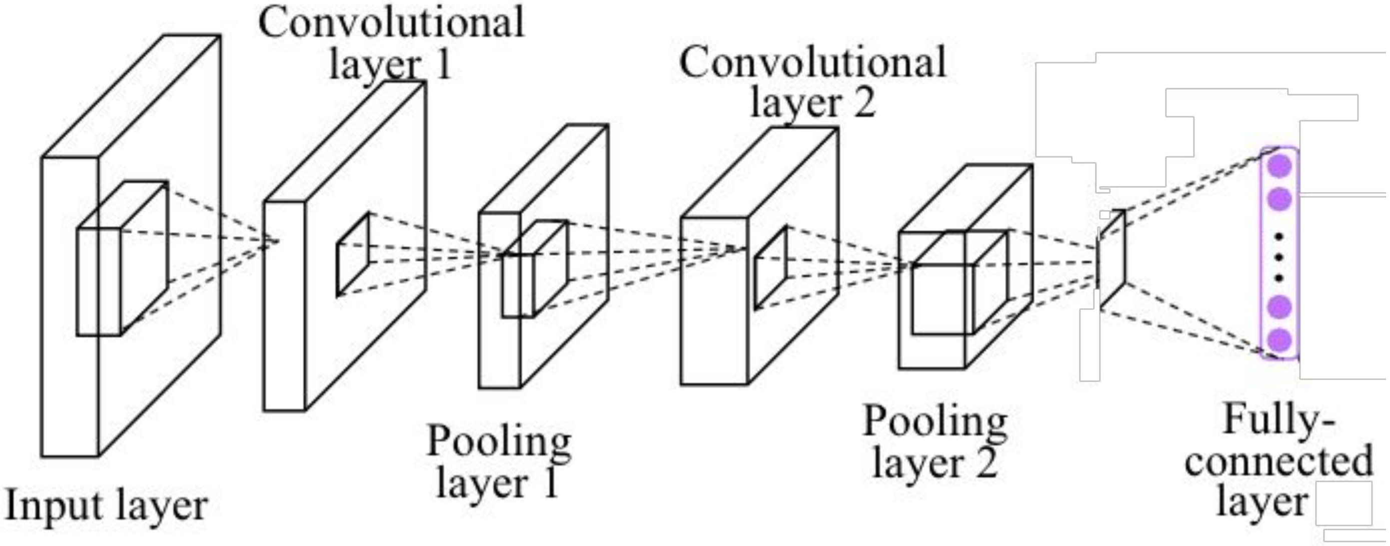


3x3x1 image

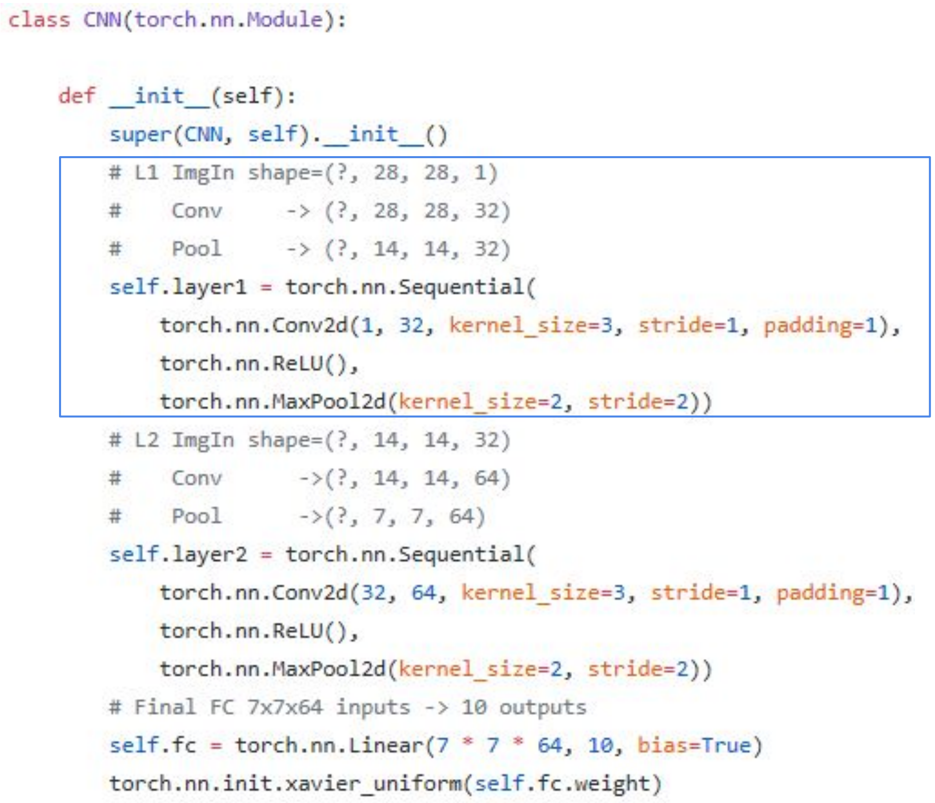
2x2x1 filter w

1 number:

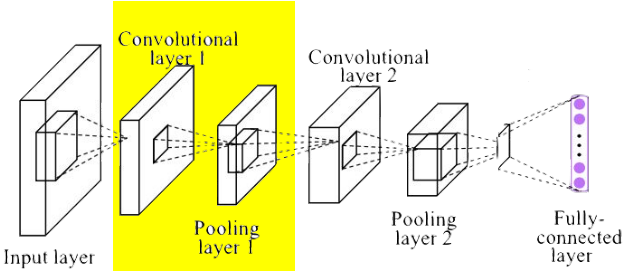
Simple CNN


Convolutional layer 1
Convolutional layer 2
Input layer
Pooling layer 1
Pooling layer 2
Fully-connected layer

Conv layer 1

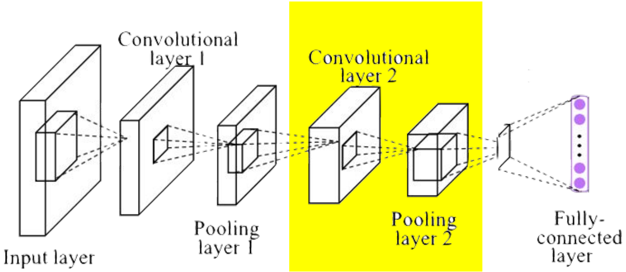


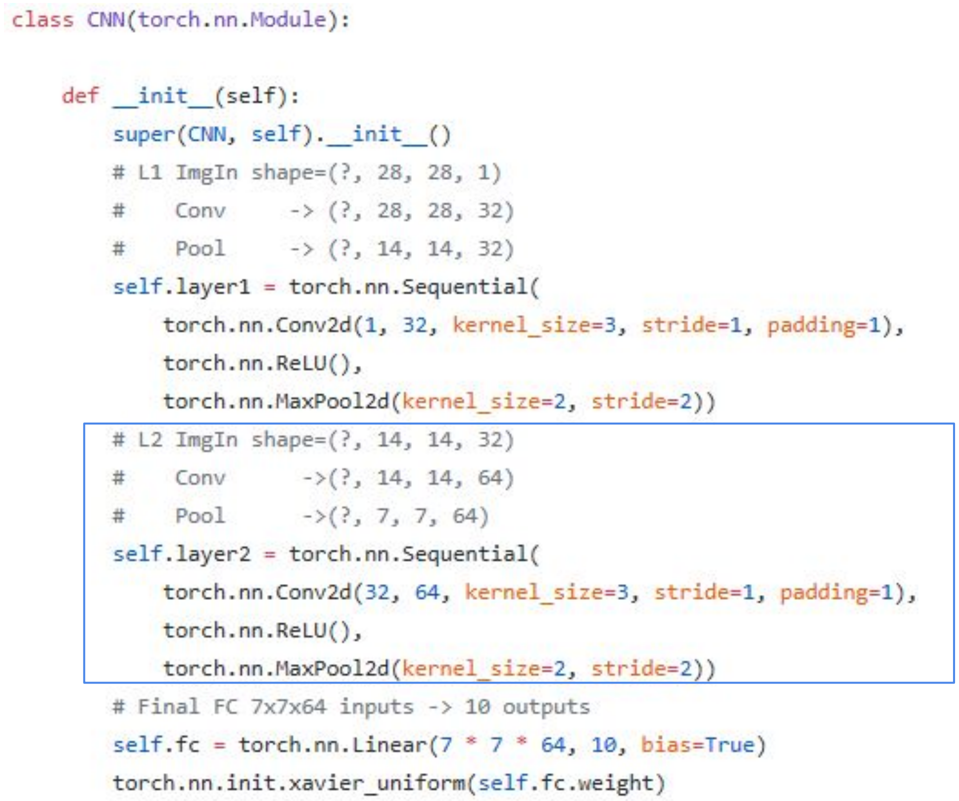
```
class CNN(torch.nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        # L1 ImgIn shape=(?, 28, 28, 1)
        #    Conv     -> (?, 28, 28, 32)
        #    Pool     -> (?, 14, 14, 32)
        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # L2 ImgIn shape=(?, 14, 14, 32)
        #    Conv      ->(?, 14, 14, 64)
        #    Pool      ->(?, 7, 7, 64)
        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # Final FC 7x7x64 inputs -> 10 outputs
        self.fc = torch.nn.Linear(7 * 7 * 64, 10, bias=True)
        torch.nn.init.xavier_uniform(self.fc.weight)
```



https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-11-1-mnist_cnn.py

Conv layer 2

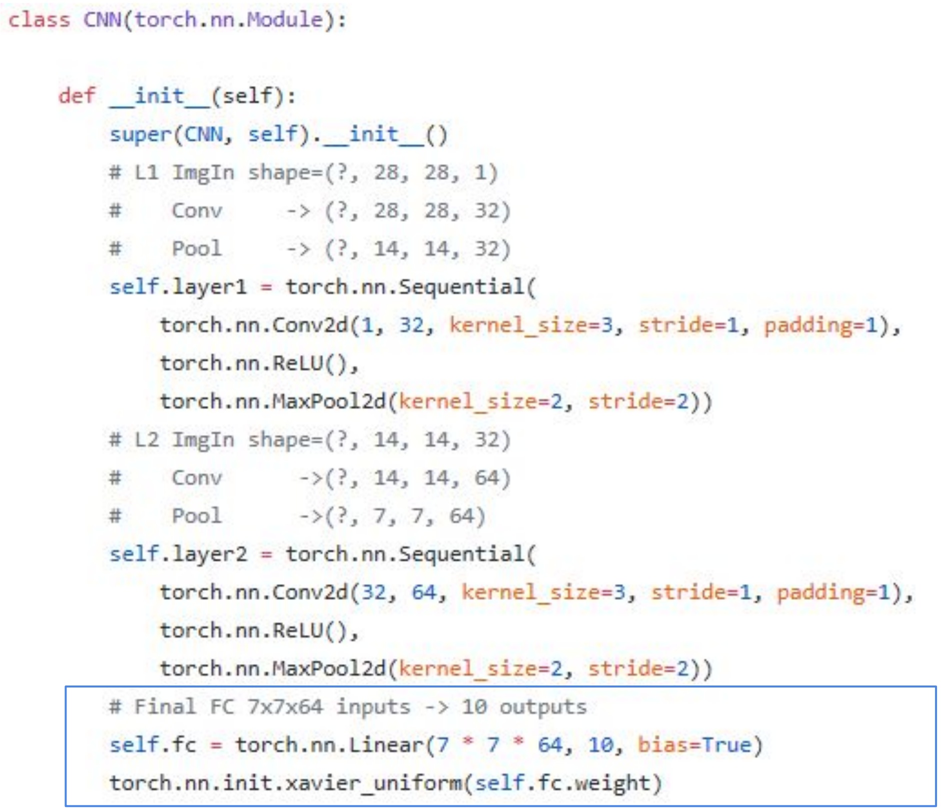




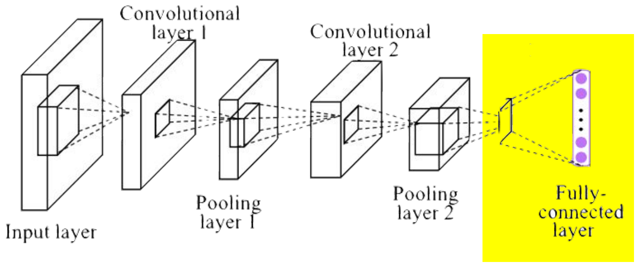
```
class CNN(torch.nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        # L1 ImgIn shape=(?, 28, 28, 1)
        #    Conv     -> (?, 28, 28, 32)
        #    Pool     -> (?, 14, 14, 32)
        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # L2 ImgIn shape=(?, 14, 14, 32)
        #    Conv      ->(?, 14, 14, 64)
        #    Pool      ->(?, 7, 7, 64)
        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # Final FC 7x7x64 inputs -> 10 outputs
        self.fc = torch.nn.Linear(7 * 7 * 64, 10, bias=True)
        torch.nn.init.xavier_uniform(self.fc.weight)
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-11-1-mnist_cnn.py

Fully Connected (FC, Dense) layer

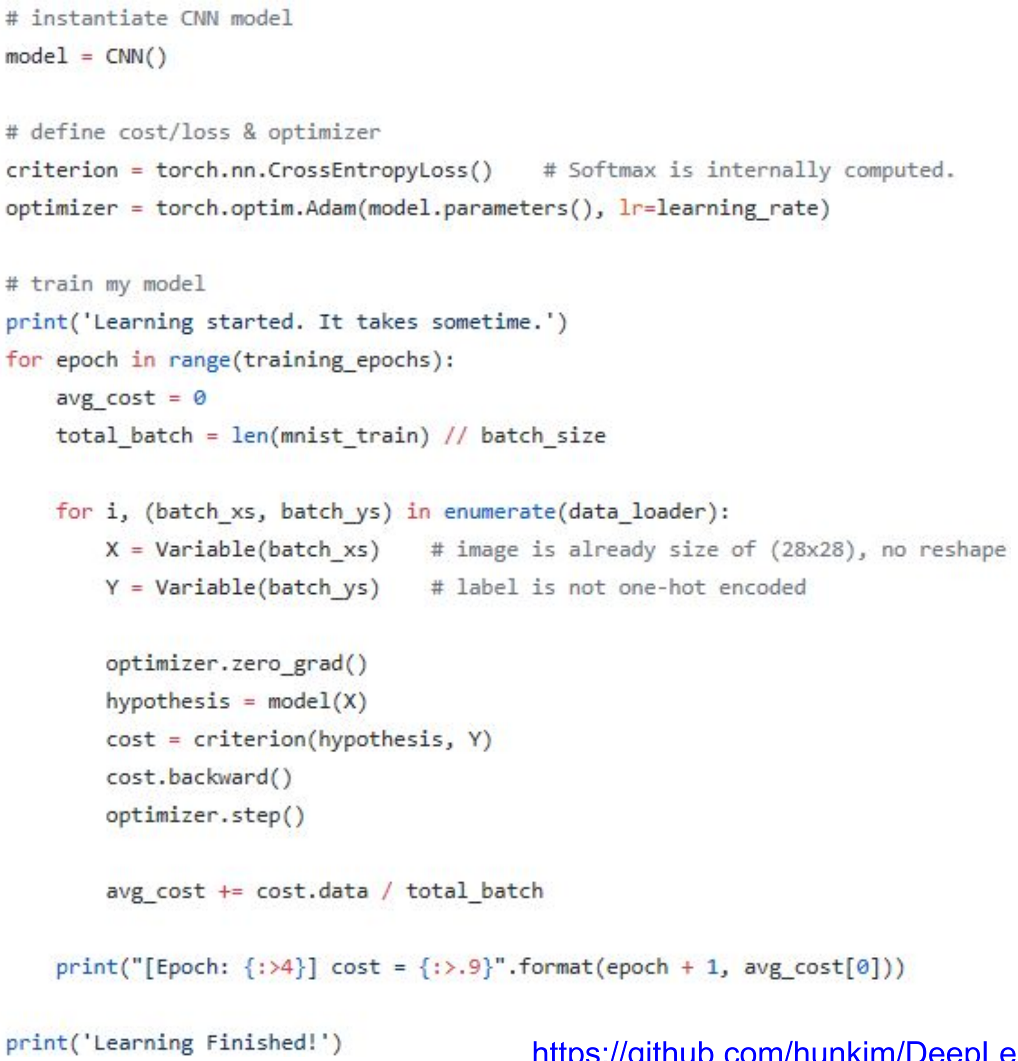


```
class CNN(torch.nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        # L1 ImgIn shape=(?, 28, 28, 1)
        #    Conv     -> (?, 28, 28, 32)
        #    Pool     -> (?, 14, 14, 32)
        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # L2 ImgIn shape=(?, 14, 14, 32)
        #    Conv      ->(?, 14, 14, 64)
        #    Pool      ->(?, 7, 7, 64)
        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # Final FC 7x7x64 inputs -> 10 outputs
        self.fc = torch.nn.Linear(7 * 7 * 64, 10, bias=True)
        torch.nn.init.xavier_uniform(self.fc.weight)
```



https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-11-1-mnist_cnn.py

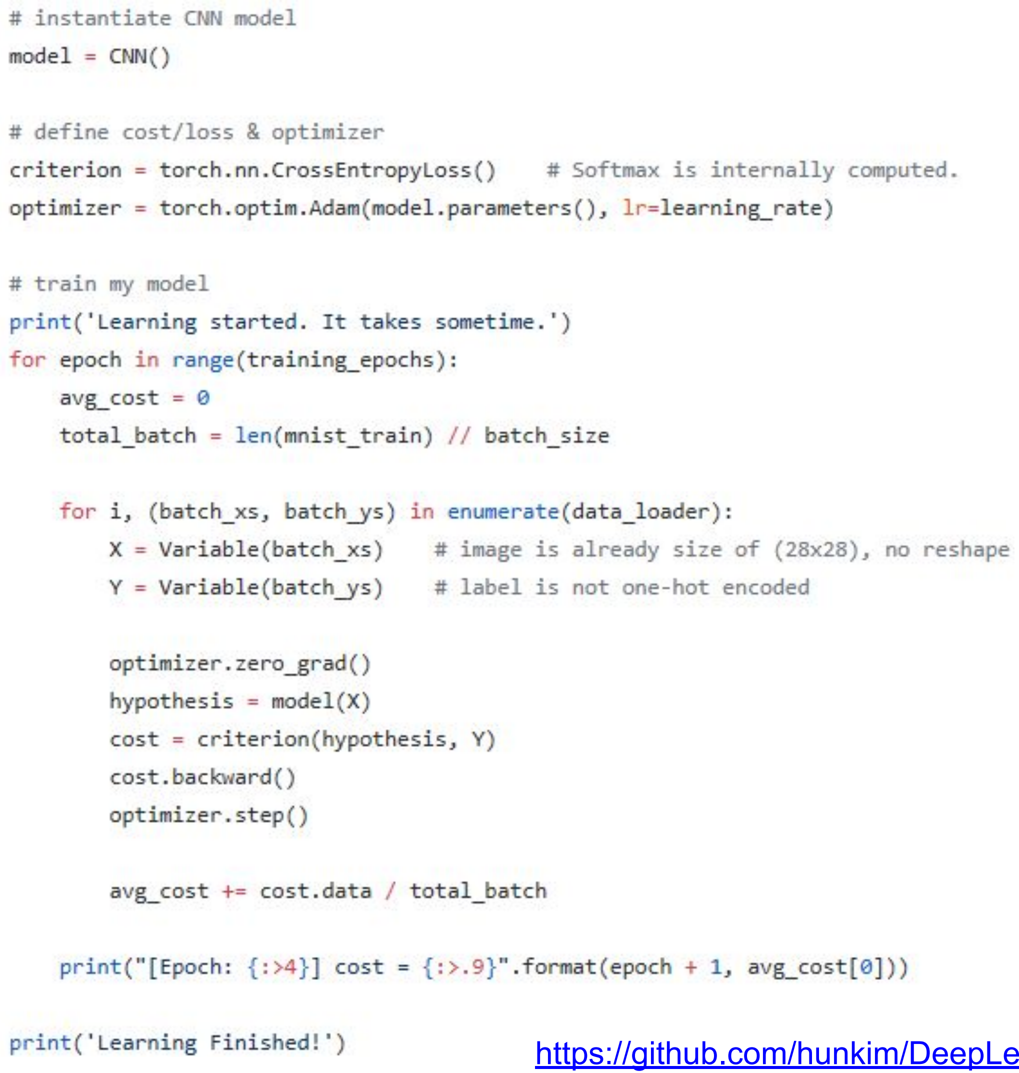
Training and Evaluation



```
# instantiate CNN model
model = CNN()

# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss()    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# train my model
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(mnist_train) // batch_size

    for i, (batch_xs, batch_ys) in enumerate(data_loader):
        X = Variable(batch_xs)    # image is already size of (28x28), no reshape
        Y = Variable(batch_ys)    # label is not one-hot encoded

        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost.data / total_batch

    print("[Epoch: {:>4}] cost = {:>.9}".format(epoch + 1, avg_cost[0]))

print('Learning Finished!')
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-11-1-mnist_cnn.py

Training and Evaluation



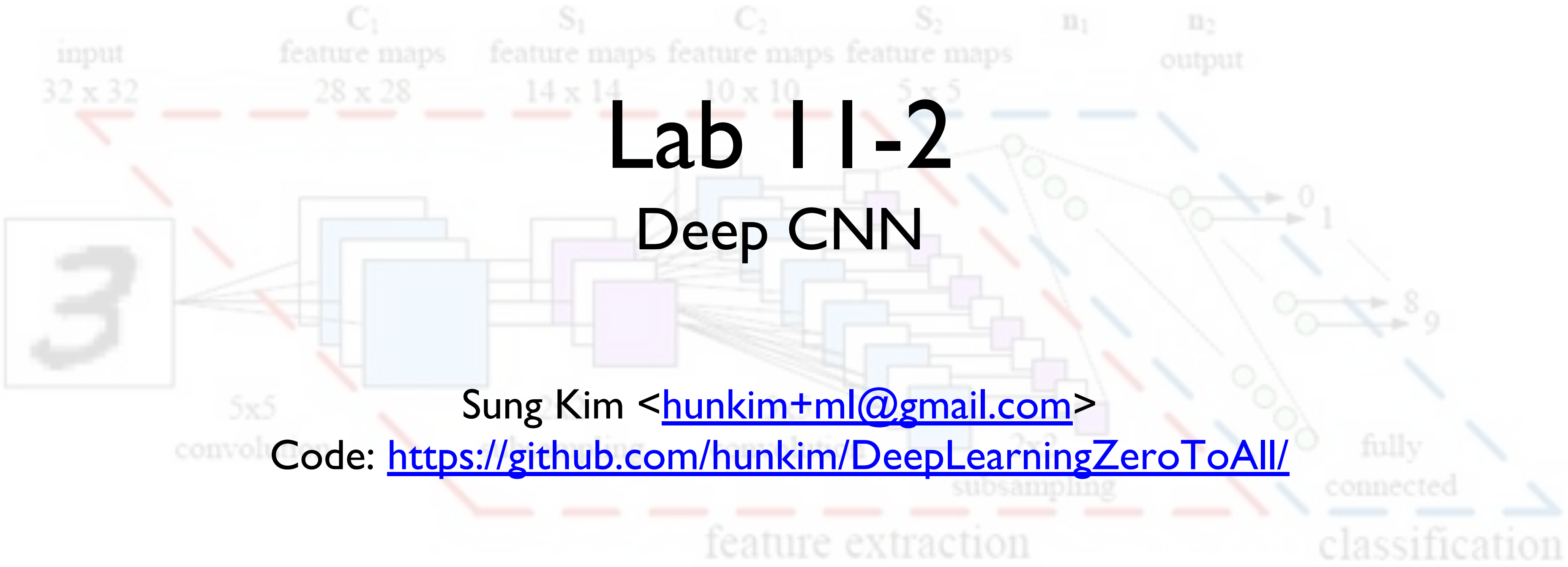
```python
# instantiate CNN model
model = CNN()

# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss()    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# train my model
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(mnist_train) // batch_size

    for i, (batch_xs, batch_ys) in enumerate(data_loader):
        X = Variable(batch_xs)    # image is already size of (28x28), no reshape
        Y = Variable(batch_ys)    # label is not one-hot encoded

        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost.data / total_batch

    print("[Epoch: {:>4}] cost = {:>.9}".format(epoch + 1, avg_cost[0]))

print('Learning Finished!')
```

```
Epoch: 0001 cost = 0.340291267
Epoch: 0002 cost = 0.090731326
Epoch: 0003 cost = 0.064477619
Epoch: 0004 cost = 0.050683064
...
Epoch: 0011 cost = 0.017758641
Epoch: 0012 cost = 0.014156652
Epoch: 0013 cost = 0.012397016
Epoch: 0014 cost = 0.010693789
Epoch: 0015 cost = 0.009469977
Learning Finished!
Accuracy: 0.9885
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-11-1-mnist_cnn.py

With PyTorch 0.2.0_2

PYTORCH



Lab 11-2

Deep CNN

Sung Kim <hunkim+ml@gmail.com>

Code: https://github.com/hunkim/DeepLearningZeroToAll/

https://github.com/hunkim/DeepLearningZeroToAll/tree/master/pytorch

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-11-2-mnist_deep_cnn.py

Deep CNN

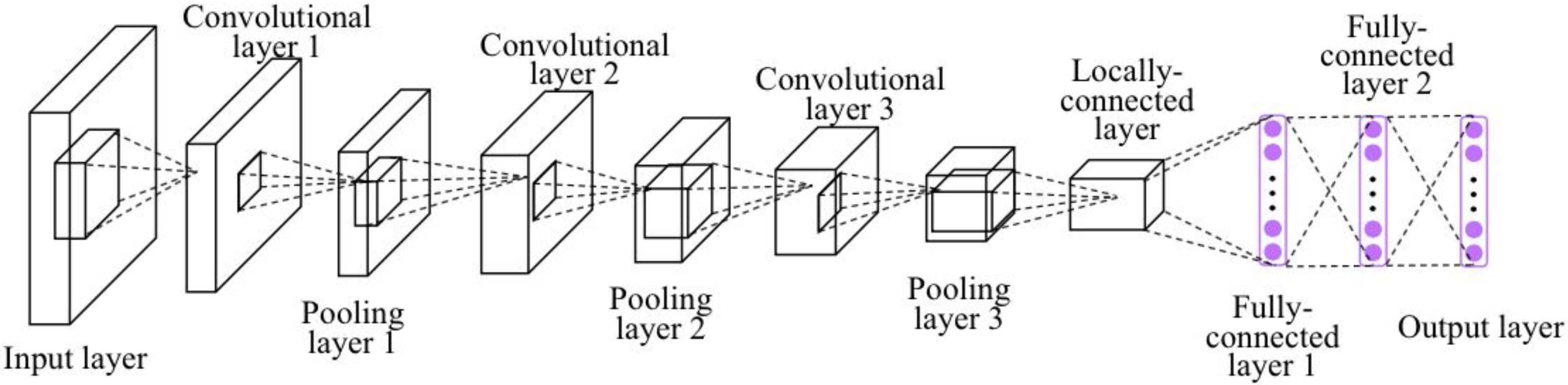
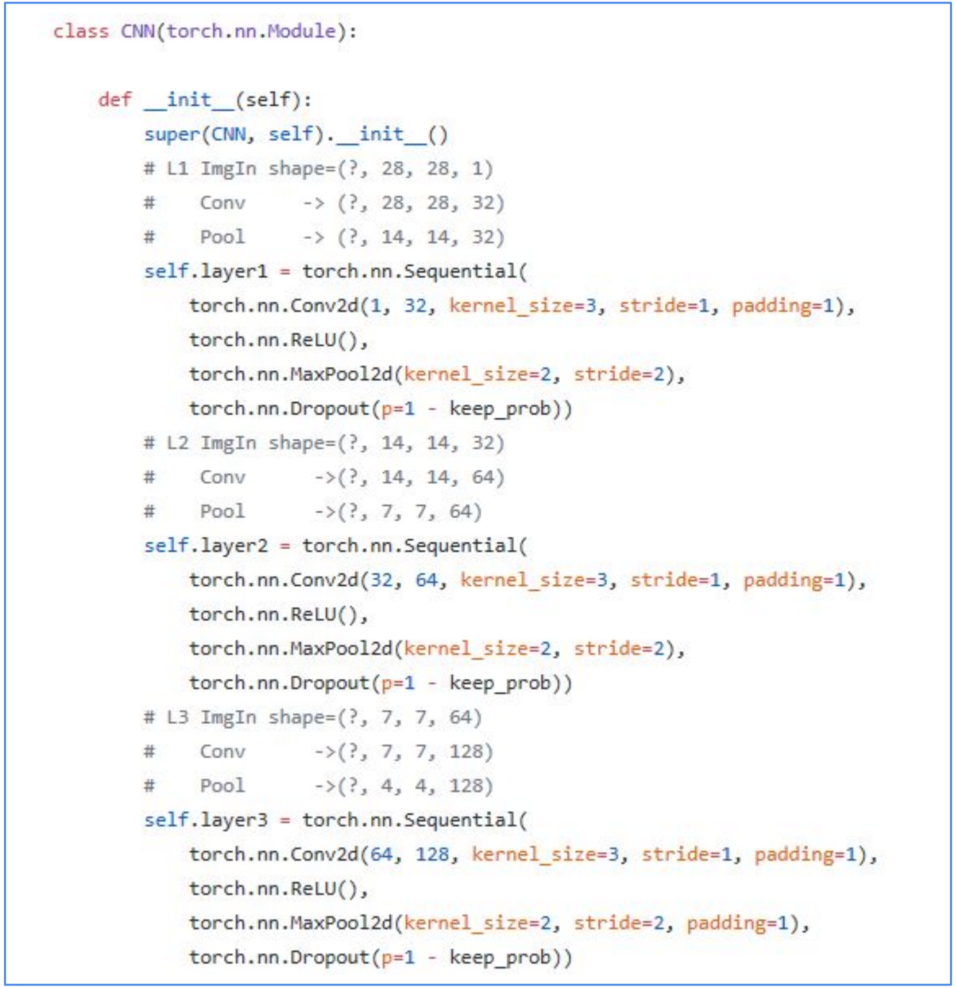
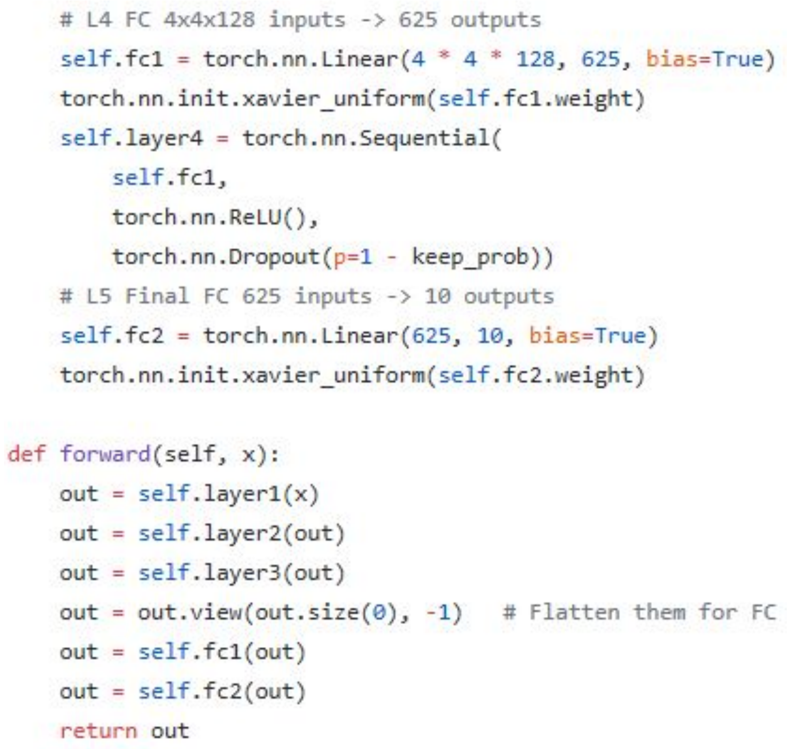


Image credit: http://personal.ie.cuhk.edu.hk/~ccloy/project_target_code/index.html

Deep CNN



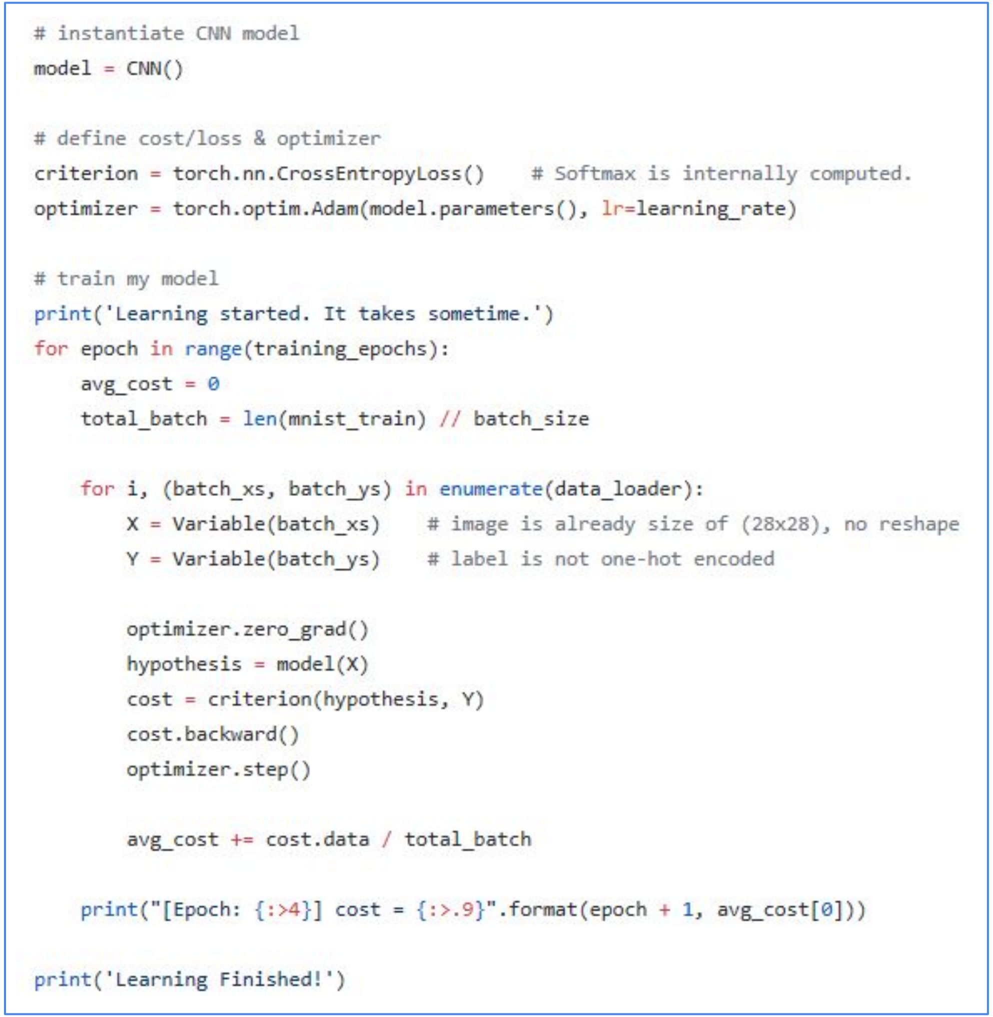
```
class CNN(torch.nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        # L1 ImgIn shape=(?, 28, 28, 1)
        #    Conv     -> (?, 28, 28, 32)
        #    Pool     -> (?, 14, 14, 32)
        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            torch.nn.Dropout(p=1 - keep_prob))
        # L2 ImgIn shape=(?, 14, 14, 32)
        #    Conv      ->(?, 14, 14, 64)
        #    Pool      ->(?, 7, 7, 64)
        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            torch.nn.Dropout(p=1 - keep_prob))
        # L3 ImgIn shape=(?, 7, 7, 64)
        #    Conv      ->(?, 7, 7, 128)
        #    Pool      ->(?, 4, 4, 128)
        self.layer3 = torch.nn.Sequential(
            torch.nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
            torch.nn.Dropout(p=1 - keep_prob))
```



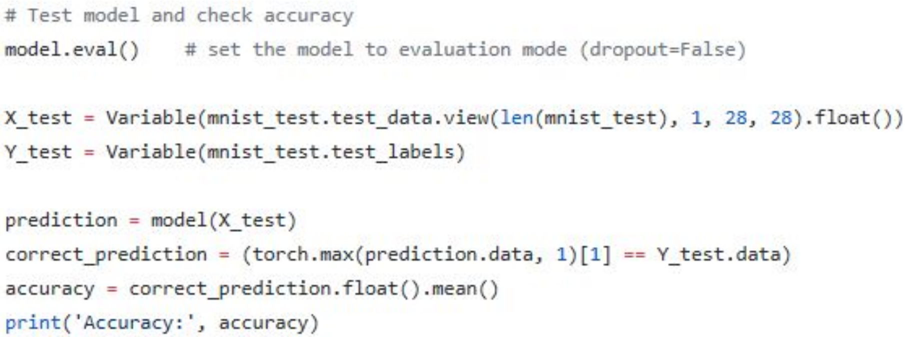
```
        # L4 FC 4x4x128 inputs -> 625 outputs
        self.fc1 = torch.nn.Linear(4 * 4 * 128, 625, bias=True)
        torch.nn.init.xavier_uniform(self.fc1.weight)
        self.layer4 = torch.nn.Sequential(
            self.fc1,
            torch.nn.ReLU(),
            torch.nn.Dropout(p=1 - keep_prob))
        # L5 Final FC 625 inputs -> 10 outputs
        self.fc2 = torch.nn.Linear(625, 10, bias=True)
        torch.nn.init.xavier_uniform(self.fc2.weight)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.view(out.size(0), -1)   # Flatten them for FC
        out = self.fc1(out)
        out = self.fc2(out)
        return out
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-11-2-mnist_deep_cnn.py

Deep CNN



```python
# instantiate CNN model
model = CNN()

# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss()    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# train my model
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(mnist_train) // batch_size

    for i, (batch_xs, batch_ys) in enumerate(data_loader):
        X = Variable(batch_xs)    # image is already size of (28x28), no reshape
        Y = Variable(batch_ys)    # label is not one-hot encoded

        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost.data / total_batch

    print("[Epoch: {:>4}] cost = {:>.9}".format(epoch + 1, avg_cost[0]))

print('Learning Finished!')
```



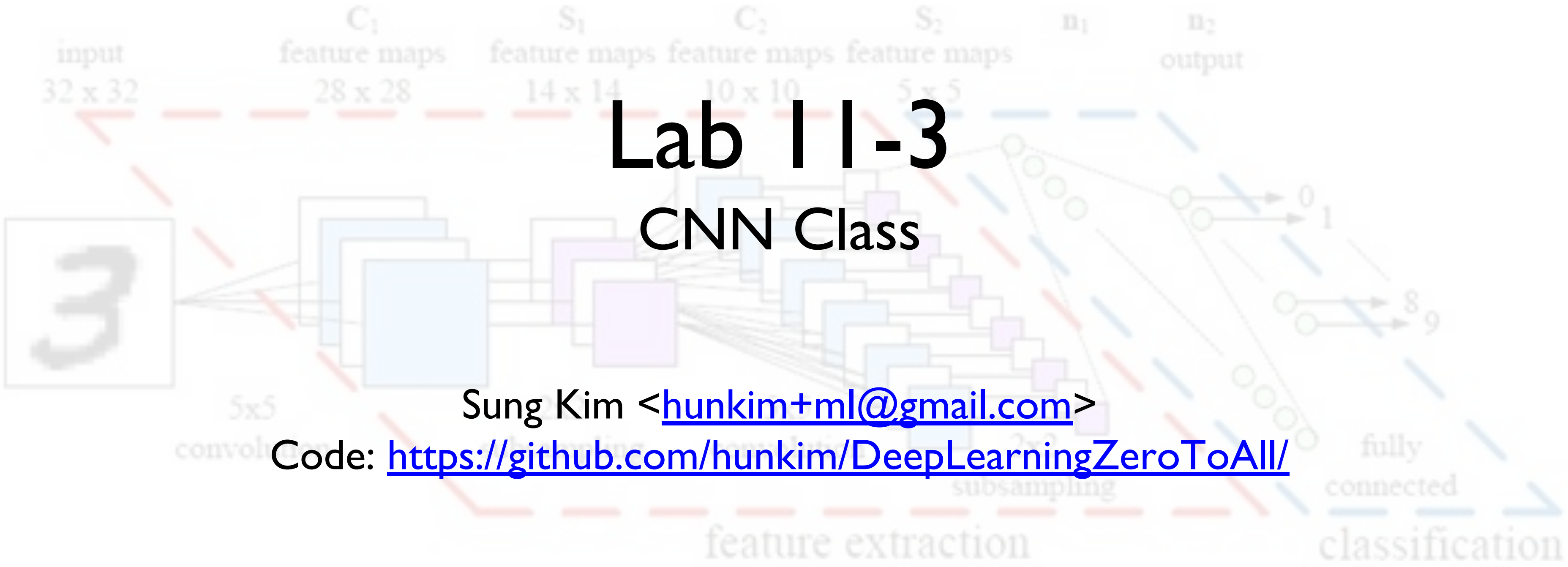
```python
# Test model and check accuracy
model.eval()    # set the model to evaluation mode (dropout=False)

X_test = Variable(mnist_test.test_data.view(len(mnist_test), 1, 28, 28).float())
Y_test = Variable(mnist_test.test_labels)

prediction = model(X_test)
correct_prediction = (torch.max(prediction.data, 1)[1] == Y_test.data)
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy)
```

Epoch: 0013 cost = 0.027188021
Epoch: 0014 cost = 0.023604777
Epoch: 0015 cost = 0.024607201
Learning Finished!

Accuracy: 0.9938

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-11-2-mnist_deep_cnn.py

With PyTorch 0.2.0_2



Lab 11-3

CNN Class

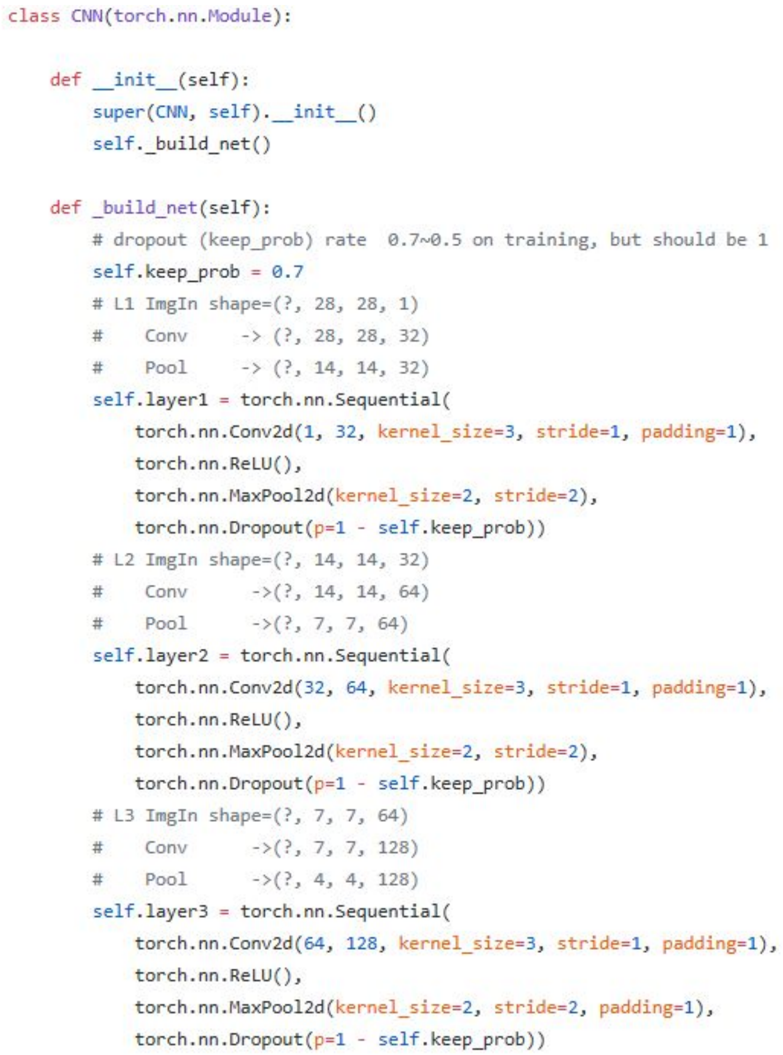
Sung Kim <hunkim+ml@gmail.com>

Code: https://github.com/hunkim/DeepLearningZeroToAll/

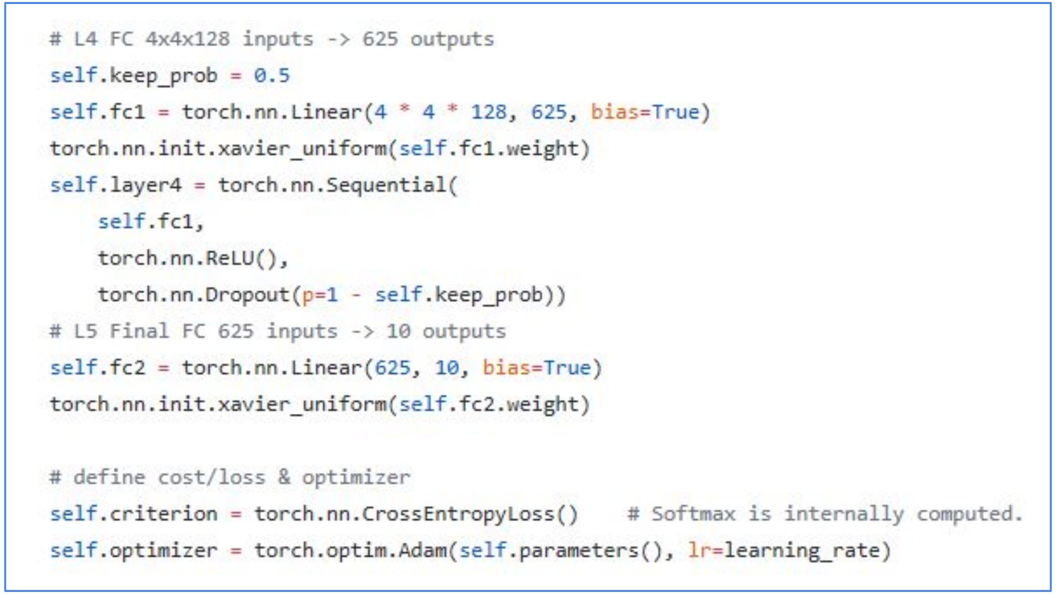
https://github.com/hunkim/DeepLearningZeroToAll/tree/master/pytorch

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-11-3-mnist_cnn_class.py

CNN
Python Class



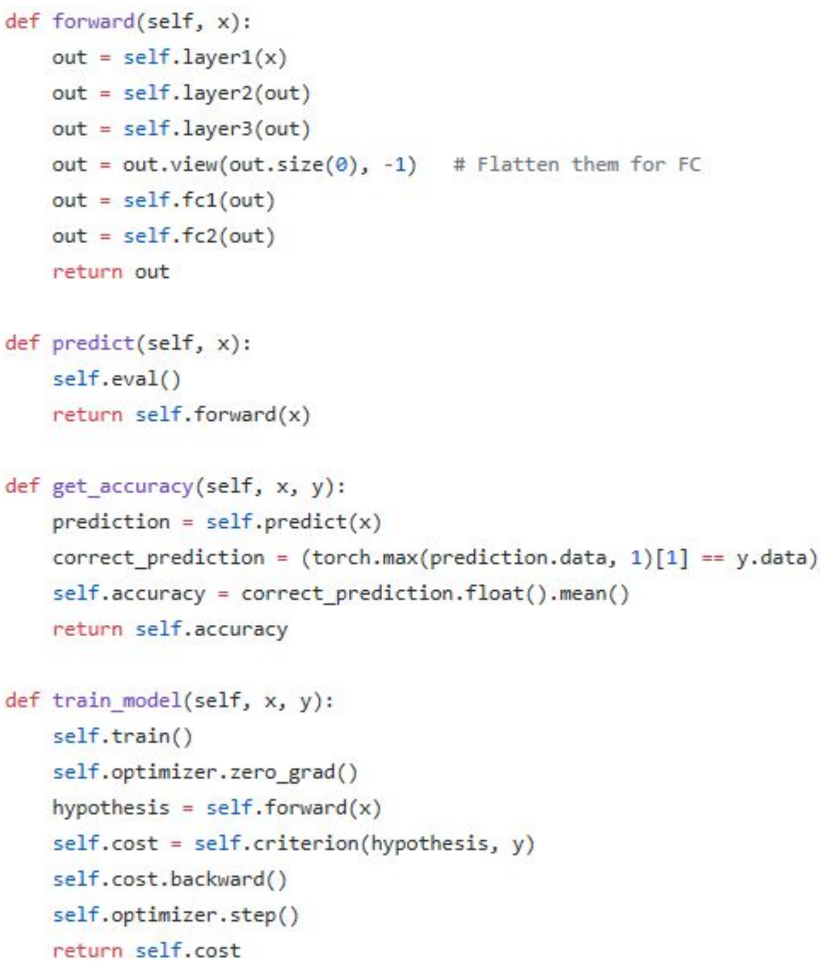
```
class CNN(torch.nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        self._build_net()

    def _build_net(self):
        # dropout (keep_prob) rate  0.7~0.5 on training, but should be 1
        self.keep_prob = 0.7
        # L1 ImgIn shape=(?, 28, 28, 1)
        #    Conv     -> (?, 28, 28, 32)
        #    Pool     -> (?, 14, 14, 32)
        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            torch.nn.Dropout(p=1 - self.keep_prob))
        # L2 ImgIn shape=(?, 14, 14, 32)
        #    Conv      ->(?, 14, 14, 64)
        #    Pool      ->(?, 7, 7, 64)
        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            torch.nn.Dropout(p=1 - self.keep_prob))
        # L3 ImgIn shape=(?, 7, 7, 64)
        #    Conv      ->(?, 7, 7, 128)
        #    Pool      ->(?, 4, 4, 128)
        self.layer3 = torch.nn.Sequential(
            torch.nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
            torch.nn.Dropout(p=1 - self.keep_prob))
```



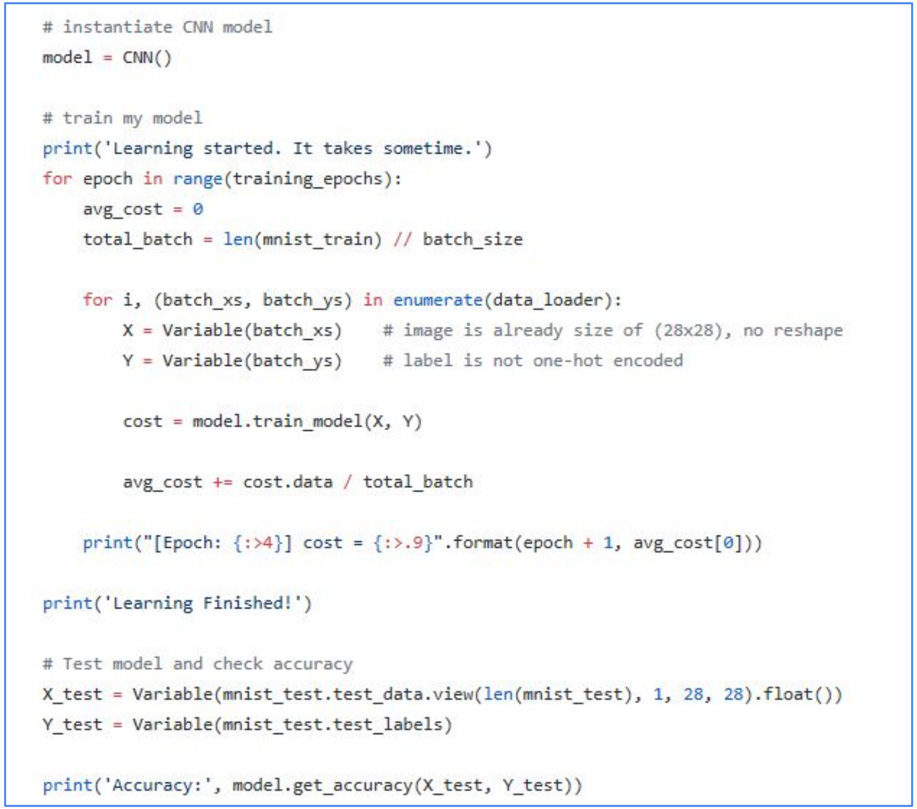
```
# L4 FC 4x4x128 inputs -> 625 outputs
self.keep_prob = 0.5
self.fc1 = torch.nn.Linear(4 * 4 * 128, 625, bias=True)
torch.nn.init.xavier_uniform(self.fc1.weight)
self.layer4 = torch.nn.Sequential(
    self.fc1,
    torch.nn.ReLU(),
    torch.nn.Dropout(p=1 - self.keep_prob))
# L5 Final FC 625 inputs -> 10 outputs
self.fc2 = torch.nn.Linear(625, 10, bias=True)
torch.nn.init.xavier_uniform(self.fc2.weight)

# define cost/loss & optimizer
self.criterion = torch.nn.CrossEntropyLoss()    # Softmax is internally computed.
self.optimizer = torch.optim.Adam(self.parameters(), lr=learning_rate)
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-11-3-mnist_cnn_class.py

CNN
Python Class



```python
def forward(self, x):
    out = self.layer1(x)
    out = self.layer2(out)
    out = self.layer3(out)
    out = out.view(out.size(0), -1)   # Flatten them for FC
    out = self.fc1(out)
    out = self.fc2(out)
    return out

def predict(self, x):
    self.eval()
    return self.forward(x)

def get_accuracy(self, x, y):
    prediction = self.predict(x)
    correct_prediction = (torch.max(prediction.data, 1)[1] == y.data)
    self.accuracy = correct_prediction.float().mean()
    return self.accuracy

def train_model(self, x, y):
    self.train()
    self.optimizer.zero_grad()
    hypothesis = self.forward(x)
    self.cost = self.criterion(hypothesis, y)
    self.cost.backward()
    self.optimizer.step()
    return self.cost
```

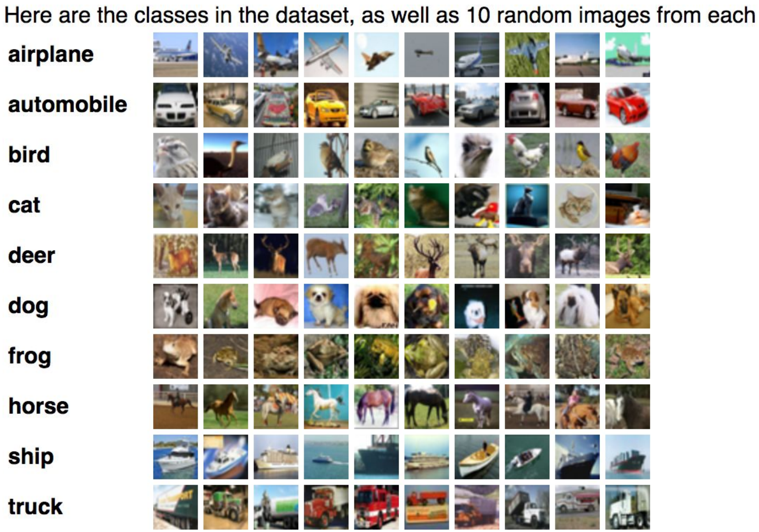


```python
# instantiate CNN model
model = CNN()

# train my model
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(mnist_train) // batch_size

    for i, (batch_xs, batch_ys) in enumerate(data_loader):
        X = Variable(batch_xs)    # image is already size of (28x28), no reshape
        Y = Variable(batch_ys)    # label is not one-hot encoded

        cost = model.train_model(X, Y)

        avg_cost += cost.data / total_batch

    print("[Epoch: {:>4}] cost = {:>.9}".format(epoch + 1, avg_cost[0]))

print('Learning Finished!')

# Test model and check accuracy
X_test = Variable(mnist_test.test_data.view(len(mnist_test), 1, 28, 28).float())
Y_test = Variable(mnist_test.test_labels)

print('Accuracy:', model.get_accuracy(X_test, Y_test))
```

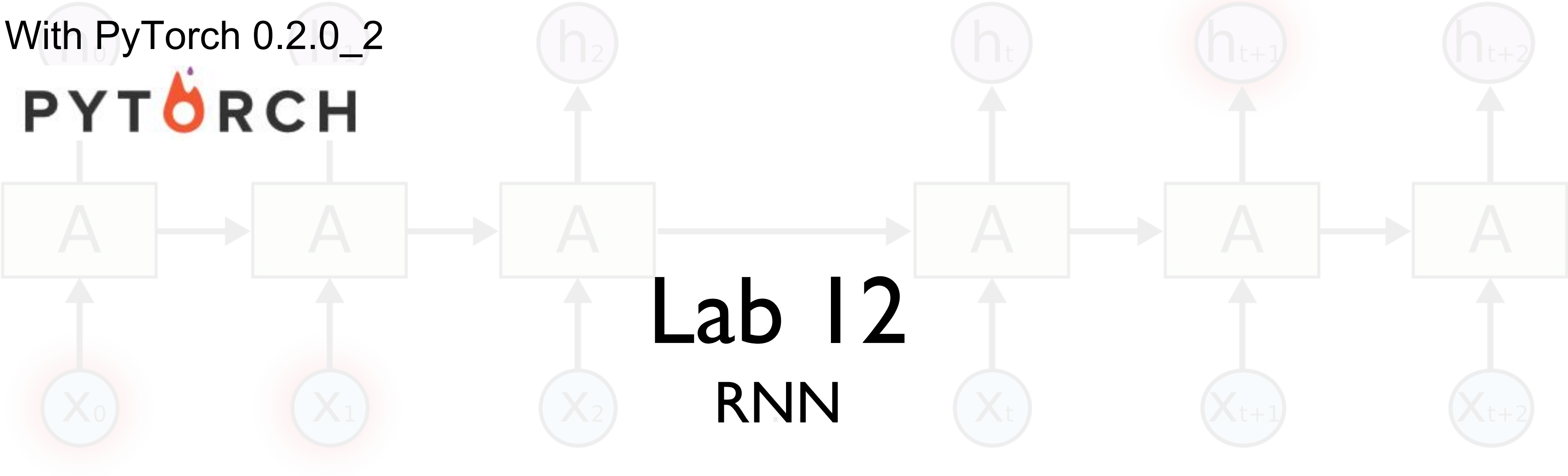
https://github.com/hunkim/DeepLearningZeroToAll/blob/master/pytorch/lab-11-3-mnist_cnn_class.py

Exercise

- Deep & Wide?
- CIFAR 10
- ImageNet


Here are the classes in the dataset, as well as 10 random images from each

With PyTorch 0.2.0_2



Lab 12

RNN

Sung Kim <hunkim+ml@gmail.com>
http://hunkim.github.io/ml/