PyTorch Zero To All 9-10

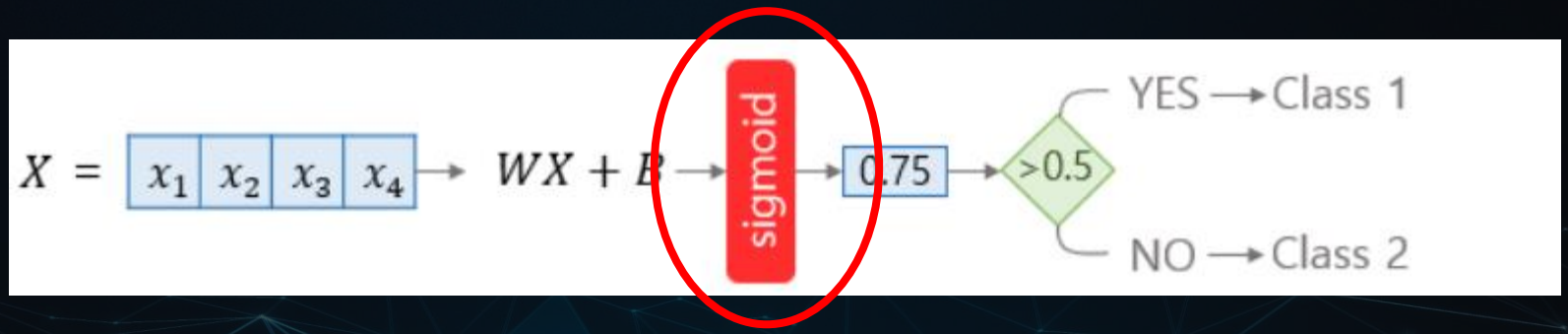
Sofmax Classifier & Basic CNN

최건우

기획
공방

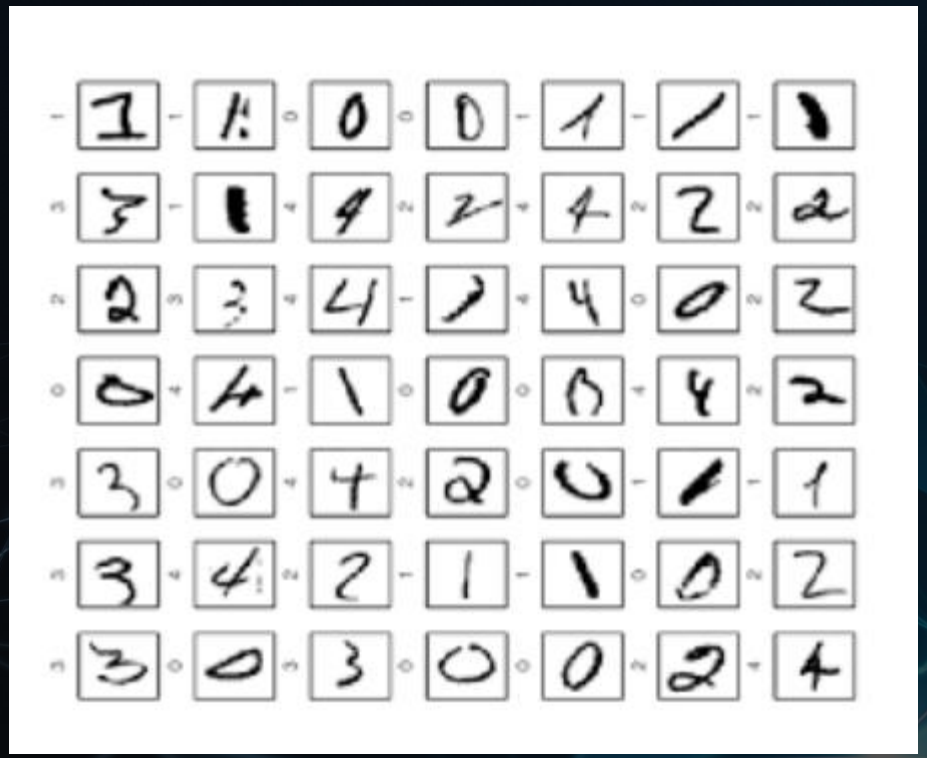
Softmax Classifier

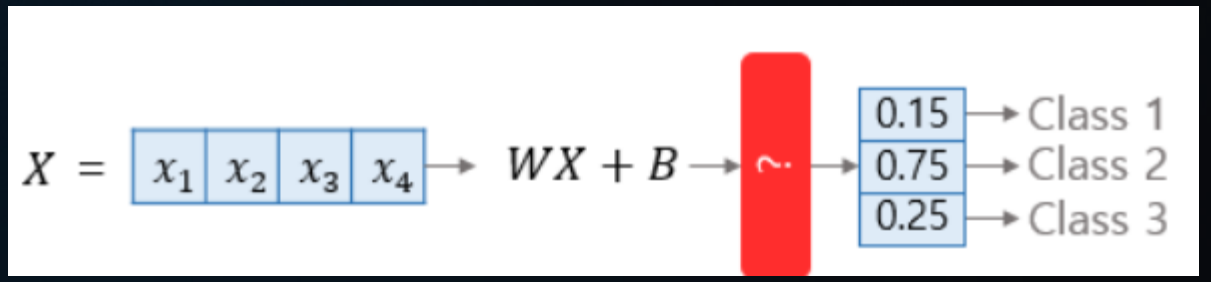
In Logistic Regression



기획
공방

Softmax Classifier



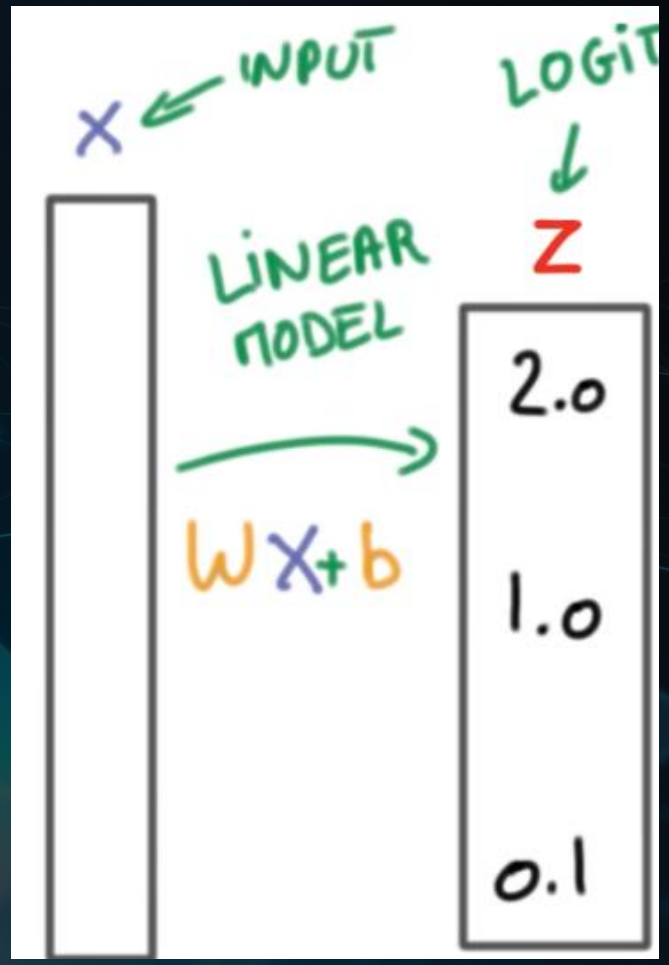


기획
공방

Softmax Classifier

Softmax function

$$softmax(z) = [\frac{e^{z_1}}{\sum_{j=1}^{3} e^{z_j}} \quad \frac{e^{z_2}}{\sum_{j=1}^{3} e^{z_j}} \quad \frac{e^{z_3}}{\sum_{j=1}^{3} e^{z_j}}]$$

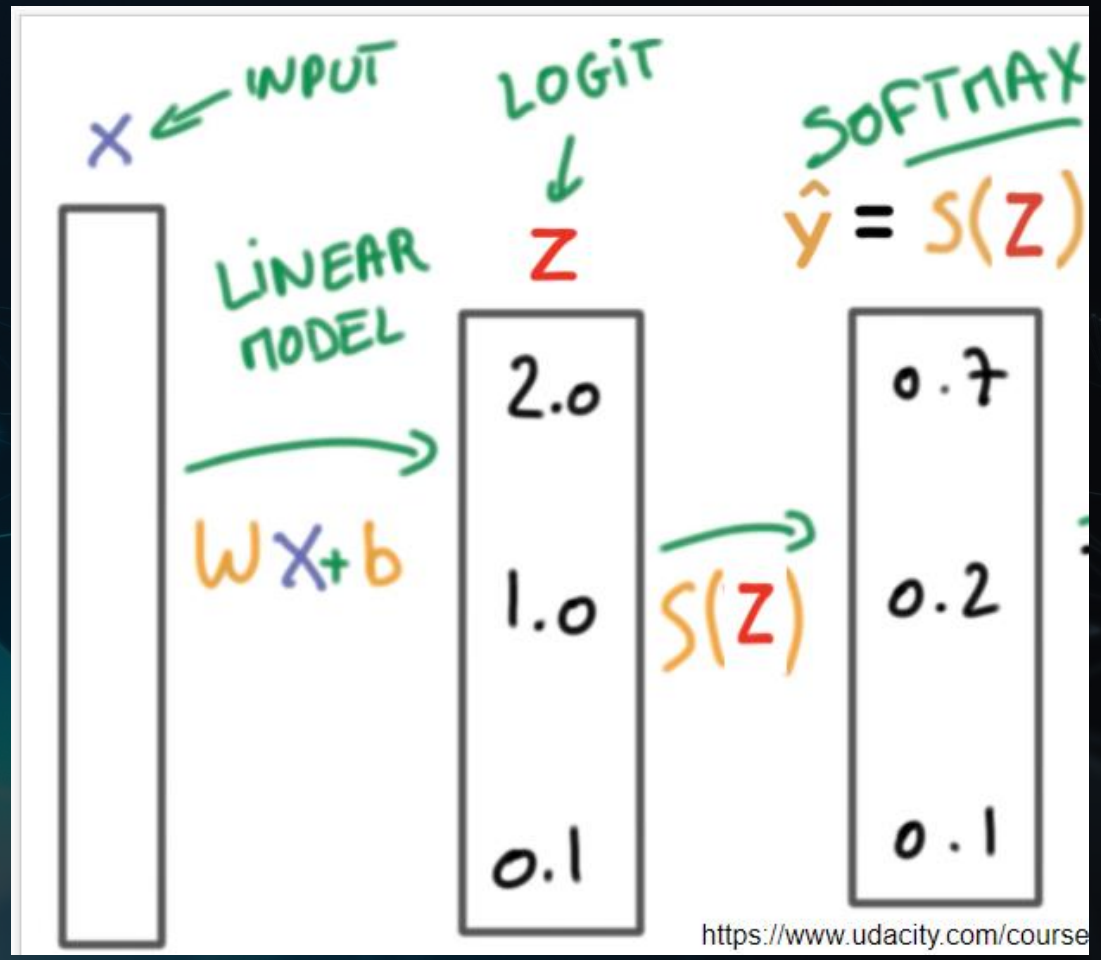


기획
공방

Softmax Classifier

Softmax function

$$softmax(z) = [\frac{e^{z_1}}{\sum_{j=1}^{3} e^{z_j}} \quad \frac{e^{z_2}}{\sum_{j=1}^{3} e^{z_j}} \quad \frac{e^{z_3}}{\sum_{j=1}^{3} e^{z_j}}]$$



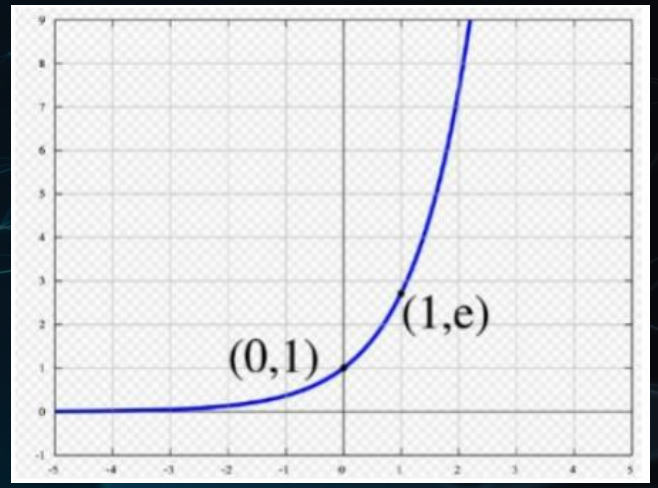
기획
공방

Softmax Classifier

Why ‘e’?

$$softmax(z) = [\frac{e^{z_1}}{\sum_{j=1}^{3} e^{z_j}} \quad \frac{e^{z_2}}{\sum_{j=1}^{3} e^{z_j}} \quad \frac{e^{z_3}}{\sum_{j=1}^{3} e^{z_j}}]$$

1. $$\frac{d}{dx}e^x = e^x$$

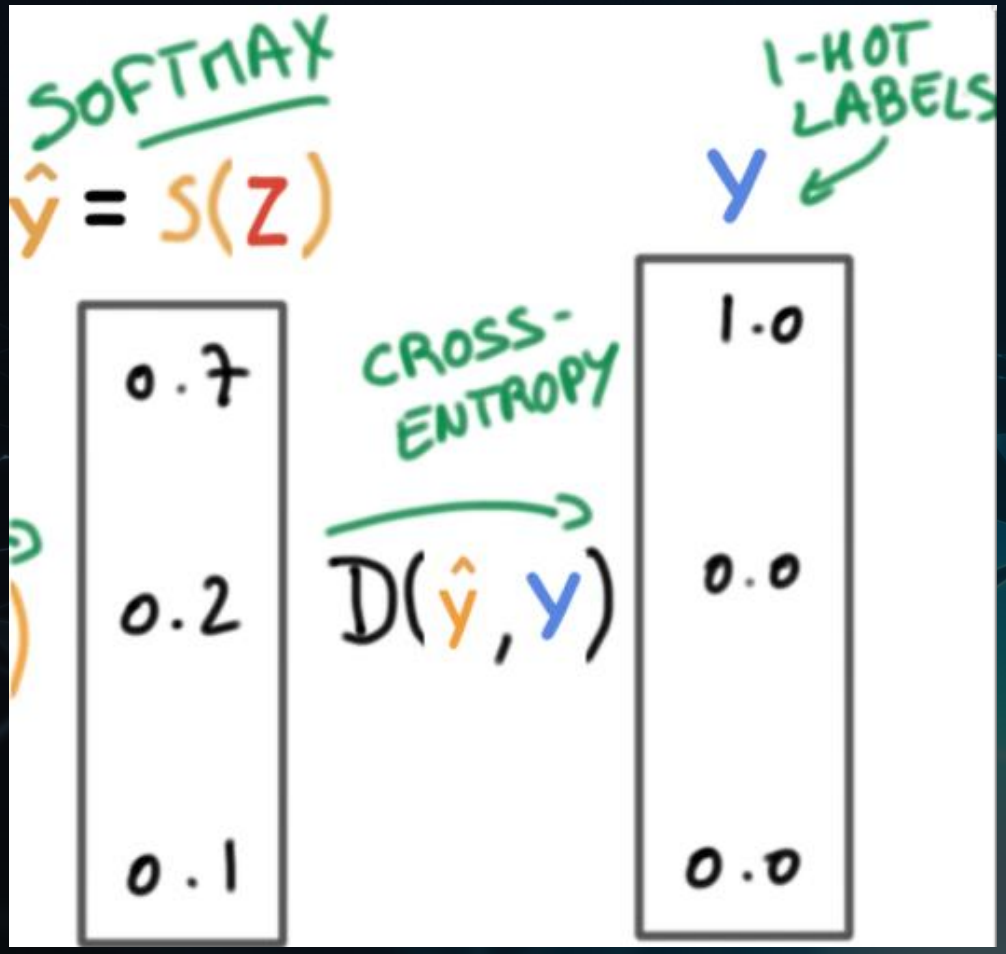
2. 

https://www.youtube.com/watch?v=K7HTd_Zgr3w&t=329s

기획
공방

Softmax Classifier

Cost function



$$cost(W) = -\sum_{j=1}^{k} y_j \, log(p_j)$$

$$cost(W) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} y_j^{(i)} \, log(p_j^{(i)})$$

기획
공방

Softmax Classifier

Logistic Regresssion Cost function

$$cost(W) = -(y\ logH(X) + (1-y)\ log(1-H(X)))$$

y->y1 (1-y)->y2
H(x)->p1 (1-H(x))->p2

$$-(y_1\ log(p_1) + y_2\ log(p_2))$$

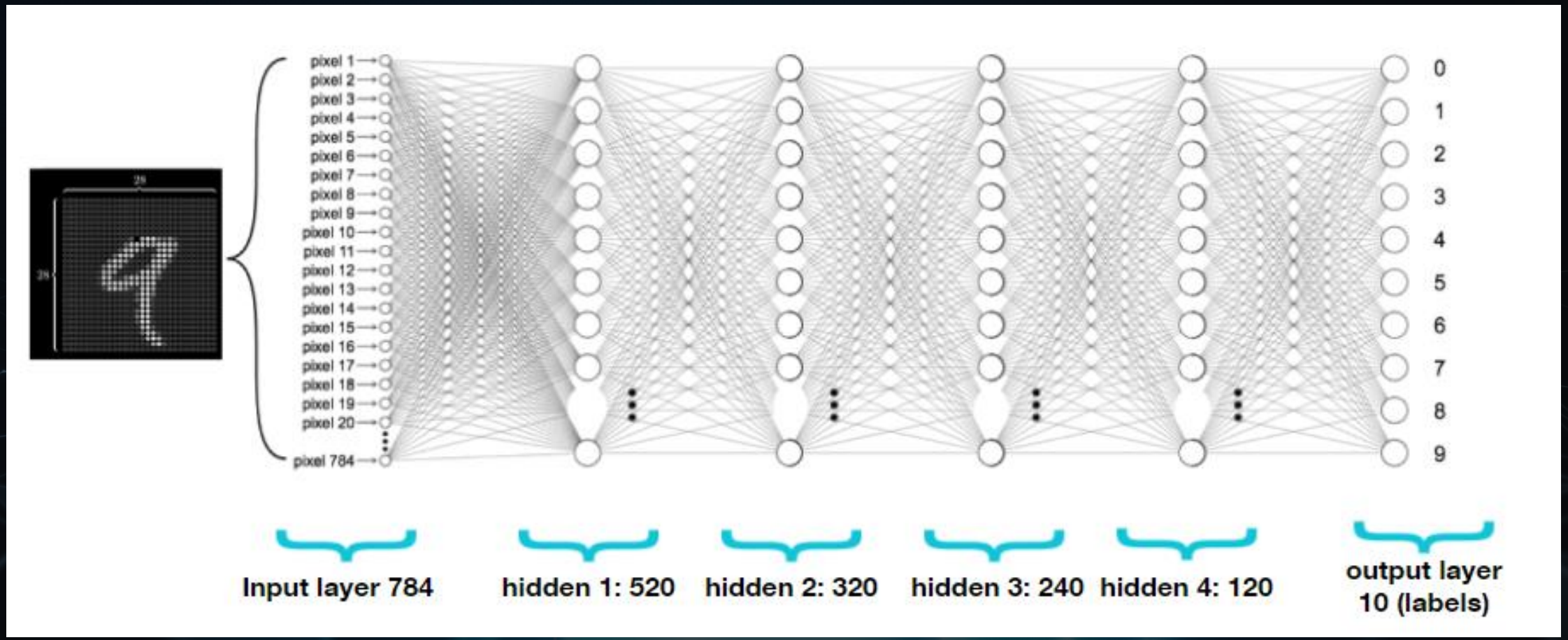
$$-(\sum_{i=1}^{2} y_i\ log\ p_i)$$

Softmax Regresssion Cost function

$$cost(W) = -\sum_{j=1}^{k} y_j\ log(p_j)$$

기획
공방

Softmax Classifier



Softmax Classifier

```python
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.l1 = nn.Linear(784, 520)
        self.l2 = nn.Linear(520, 320)
        self.l3 = nn.Linear(320, 240)
        self.l4 = nn.Linear(240, 120)
        self.l5 = nn.Linear(120, 10)

    def forward(self, x):
        x = x.view(-1, 784)  # Flatten the data (n, 1, 28, 28)-> (n, 784)
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = F.relu(self.l3(x))
        x = F.relu(self.l4(x))
        return self.l5(x)
```

```python
criterion = nn.CrossEntropyLoss()

'''
for batch_idx, (data, target) in enumerate(train_loader):
    data, target = Variable(data), Variable(target)
    optimizer.zero_grad()
    output = model(data)
    loss = criterion(output, target)
    loss.backward()
    optimizer.step()
```

Softmax Classifier

```python
def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} | Batch Status: {}/{} ({:.0f}%) | Loss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))


def test():
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        output = model(data)
        # sum up batch loss
        test_loss += criterion(output, target).item()
        # get the index of the max
        pred = output.data.max(1, keepdim=True)[1]
        correct += pred.eq(target.data.view_as(pred)).cpu().sum()

    test_loss /= len(test_loader.dataset)
    print(f'============================₩nTest set: Average loss: {test_loss:.4f}, Accuracy: {correct}/{len(test_loader.dataset
          f'({100. * correct / len(test_loader.dataset):.0f}%)')


if __name__ == '__main__':
    since = time.time()
    for epoch in range(1, 10):
        epoch_start = time.time()
        train(epoch)
        m, s = divmod(time.time() - epoch_start, 60)
        print(f'Training time: {m:.0f}m {s:.0f}s')
        test()
        m, s = divmod(time.time() - epoch_start, 60)
        print(f'Testing time: {m:.0f}m {s:.0f}s')

    m, s = divmod(time.time() - since, 60)
    print(f'Total Time: {m:.0f}m {s:.0f}s₩nModel was trained on {device}!')
```

```
Test set: Average loss: 0.0016, Accuracy: 9691/10000 (97%)
Testing time: 0m 22s
Total Time: 3m 39s
Model was trained on cpu!
```

기획
공방

Softmax Classifier

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.l1 = nn.Linear(784, 650)
        self.l2 = nn.Linear(650, 530)
        self.l3 = nn.Linear(530, 420)
        self.l4 = nn.Linear(420, 350)
        self.l5 = nn.Linear(350, 290)
        self.l6 = nn.Linear(290, 200)
        self.l7 = nn.Linear(200, 150)
        self.l8 = nn.Linear(150, 80)
        self.l9 = nn.Linear(80, 10)

    def forward(self, x):
        x = x.view(-1, 784)  # Flatten the data (n, 1, 28, 28)-> (n, 784)
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = F.relu(self.l3(x))
        x = F.relu(self.l4(x))
        x = F.relu(self.l5(x))
        x = F.relu(self.l6(x))
        x = F.relu(self.l7(x))
        x = F.relu(self.l8(x))
        return self.l9(x)
```

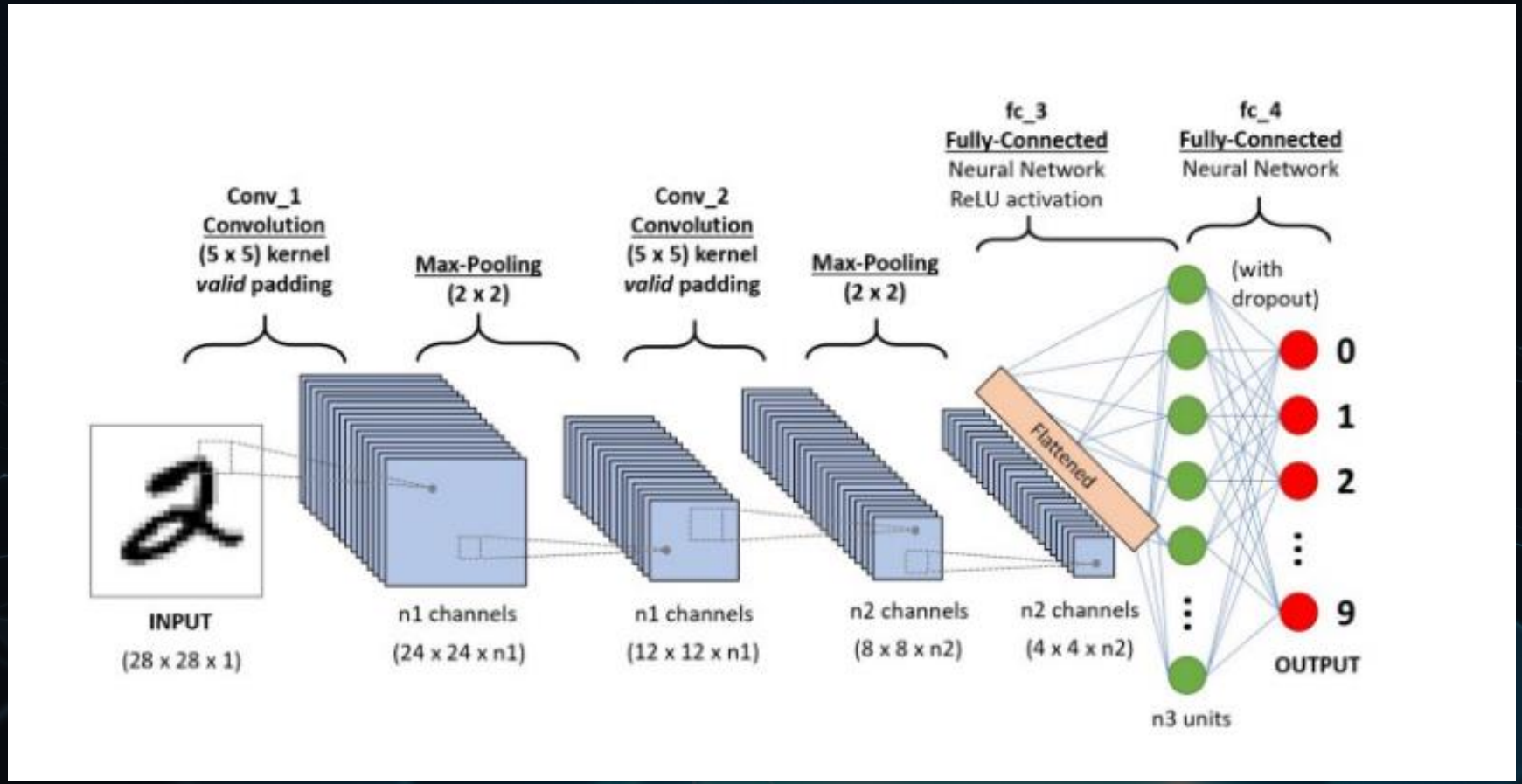
```
Test set: Average loss: 0.0361, Accuracy: 1135/10000 (11%)
Testing time: 0m 30s
Total Time: 5m 17s
Model was trained on cpu!
```

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.l1 = nn.Linear(784, 420)
        self.l2 = nn.Linear(420, 120)
        self.l3 = nn.Linear(120, 10)

    def forward(self, x):
        x = x.view(-1, 784)  # Flatten the data (n, 1, 28, 28)-> (n, 784)
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        return self.l3(x)
```

```
Test set: Average loss: 0.0022, Accuracy: 9574/10000 (96%)
Testing time: 0m 28s
Total Time: 2m 18s
Model was trained on cpu!
```

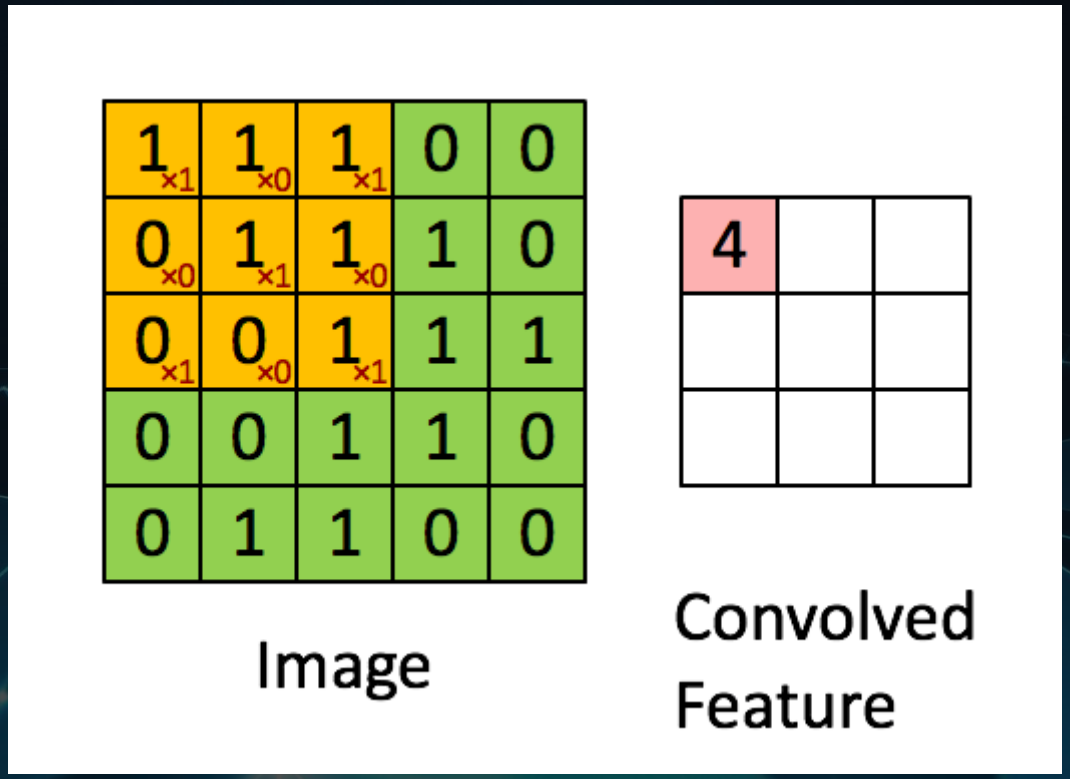
기획
공방

Basic CNN



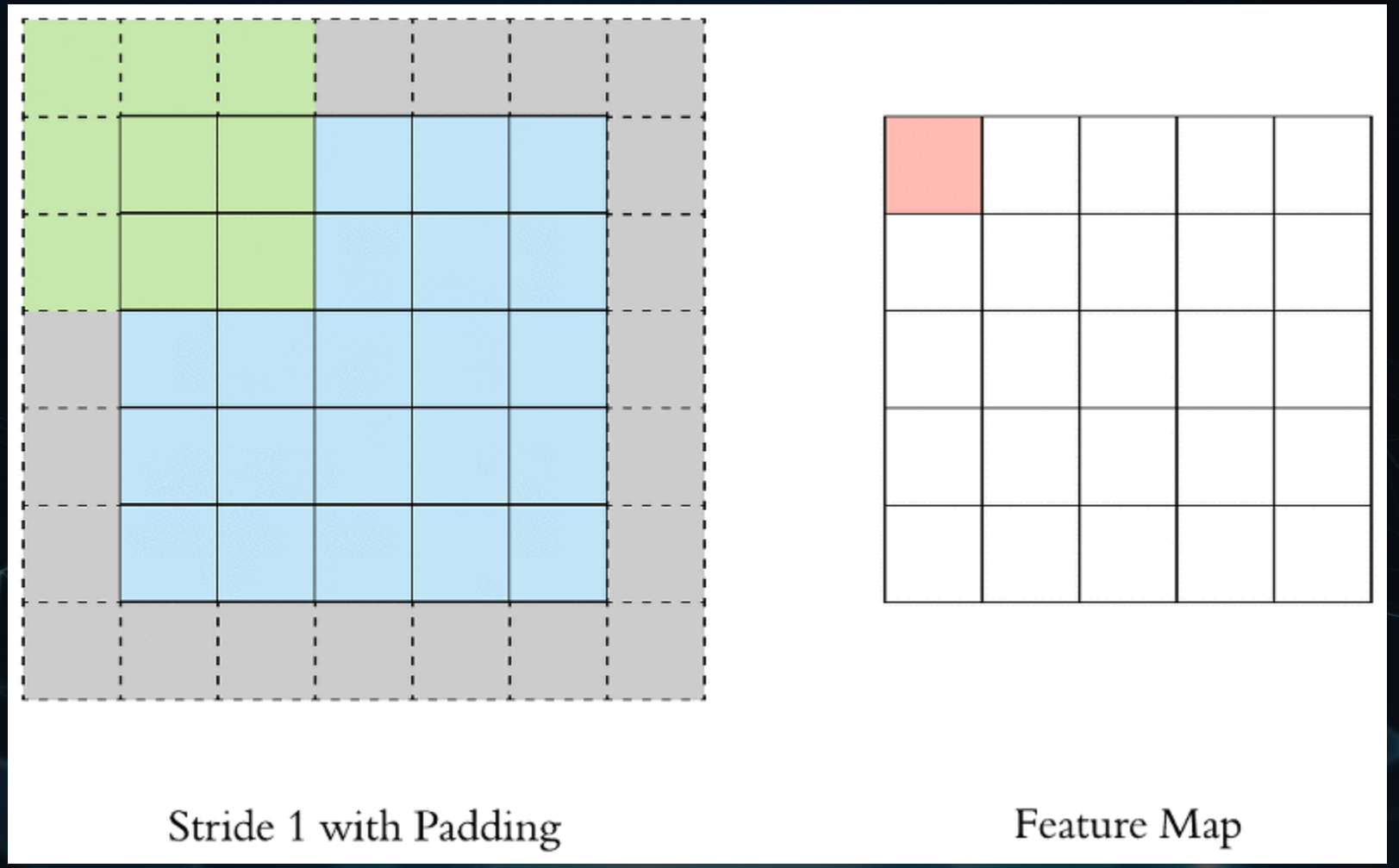
기획
공방

Basic CNN



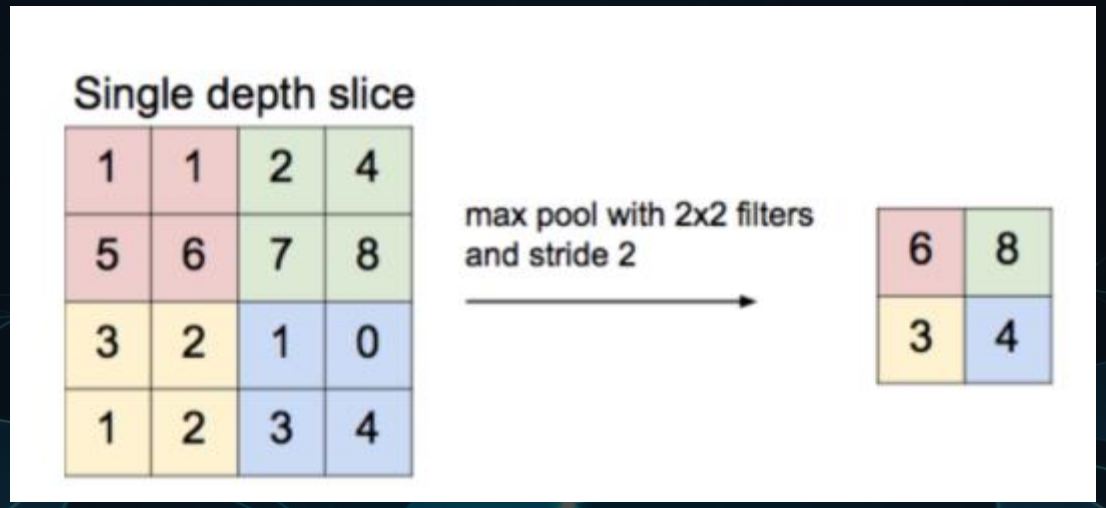
기획
공방

Basic CNN



기획
공방

Basic CNN



기획
공방

Basic CNN

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(320, 10)

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1)  # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```

input:1x28x28

10x24x24 ->10x12x12

20x8x8 ->20x4x4=320

Basic CNN

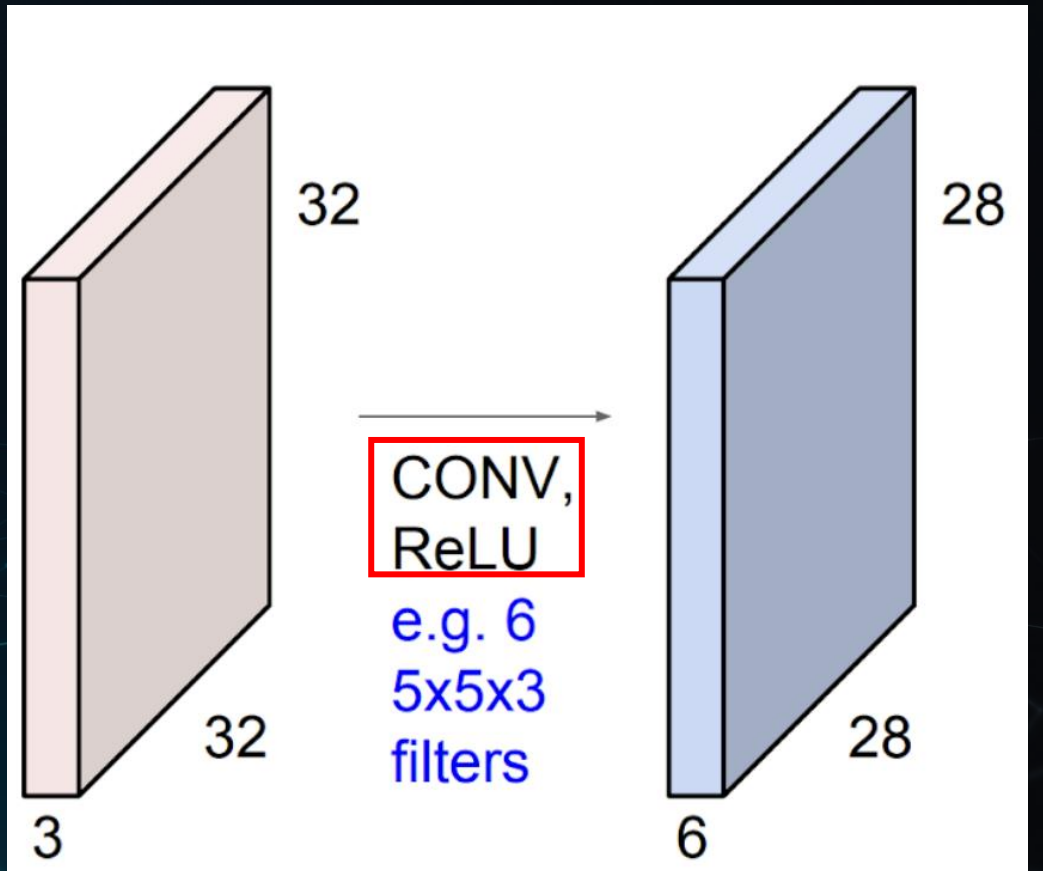
```python
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(???, 10)

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1)  # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```

```
RuntimeError: mat1 and mat2 shapes cannot be multiplied (64x320 and 100x10)
```

기획
공방

Basic CNN

```python
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(320, 10)

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1)  # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```



MaxPool(Relu(x)) = Relu(MaxPool(x))

기획
공방

Basic CNN

```
model = Net()

optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)


def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))


def test():
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        data, target = Variable(data, volatile=True), Variable(target)
        output = model(data)
        # sum up batch loss
        test_loss += F.nll_loss(output, target, size_average=False).data
        # get the index of the max log-probability
        pred = output.data.max(1, keepdim=True)[1]
        correct += pred.eq(target.data.view_as(pred)).cpu().sum()

    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))


for epoch in range(1, 10):
    train(epoch)
    test()
```

```
Test set: Average loss: 0.0576, Accuracy: 9814/10000 (98%)
```

기획
공방

D-ai-ving

감사합니다!